The Requirements Analysis & Design for a Clinical Information System: A Formal Approach

by

Jeremy David Hassé Holland

A PhD Thesis

submitted to

City University,

The Department of Systems Science

January 1995

To Emma,
to my parents,
and to my surprise.

Contents

Table of Figures

Acknowledgements

This work would not have been completed were it not for the help and support of many friends and colleagues. I extend my gratitude to all of these: there are a number which I would like to thank particularly, however. Peter Sönksen for being a tolerant and supportive employer for three and a half years of the work's germination. Ewart Carson and Bernie Cohen for providing valuable guidance and supervision during the course of the project, and for reviewing the thesis. David Russell-Jones and Jake Powrie for being 'tame clinicians' willing to give much time to help me understand the rudiments of medical care and practice. Stella Harding for interesting and useful sociological discussions during my time as her 'guinea-pig analyst'. My father for painstakingly reviewing the thesis and suggesting many useful modifications. My brother and Jerry Barnes for illuminating me in the subject of analytical methods used by civil engineers. Logica, my current employers, for letting me use their facilities during the latter stages of the 'write-up'. Finally, I would like to thank Steve Schuman for giving me so much of his time, wisdom, energy, and inspiration without which this thesis would almost certainly never have been written.

Declaration

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

Abstract

Following a number of recent far-reaching reforms to the UK NHS, St Thomas' Hospital (where this work was based) introduced a management structure based on the 'Clinical Directorate'. In order to lessen the increased workload commensurate with this measure, it was decided at St Thomas' that a new type of information system - the Directorate Information System or DIS - would introduced. This system was to 'support the business of the clinical directorate'. As part of the DIS project, a small study was set up to investigate the problems associated with the introduction of such an information system, and to suggest a design. This thesis reports on the study.

The design of information systems in general, and clinical information systems in particular, seems to be an extremely difficult endeavour: many systems development projects end in failure. It is widely considered that the problems lie in inadequate requirements analysis and specification: consequently it was here that the project concentrated most of its efforts.

It was recognised that when in use, the terms, quantities, and entities stored and displayed by an information system are interpreted by its users as terms, quantities, and entities in the organisation that is being supported (also called the domain in the thesis). This is perhaps the fundamental requirement of an information system: that it represents the organisation and processes it is to support.

To assess the degree to which a design satisfies this requirement entails the development and use of three descriptions, or theories. The first is the theory of the domain; the second is a theory, or specification, of the proposed information system; the third is a theory of the way in which the information system is interpreted into the domain - this is called the interaction theory and is a composition of the first two theories. By inspecting the interaction theory inadequacies in the representation of the domain by the information system can be identified and, if necessary, rectified. There are four ways in which we are encouraged to modify information system designs so that they more accurately reflect the behaviour of the domain. These are called the four developmental motives. Through the use of a well constructed interaction theory, and guided by the desire for system simplicity on one hand and the four developmental motives on the other, an improved information system design can be engineered.

For an interaction theory to be constructed and provide useful insight, both the domain theory and the information system specification must be semantically rich. Conventional analysis notations are inadequate for the task: mathematics (in this case set theory) is needed to represent and explore the domain, the information system, and the interpretation of the latter into the former.

The construction of a good domain theory is the hardest part of the process. Representing the organisation as it is perceived by workers (in this case clinicians) as a set theoretic construction is fraught with difficulties. However, the judicious use of an adaptation of the scientific method means that we can have increased confidence that the resulting description of the organisation is a reasonable one and is not merely a statement of the analyst's preconceptions and prejudices.

The thesis describes in more detail the background to the project, the use of the scientific method to derive a domain theory, the construction of interaction theories, and the engineering of information systems through the use of the four developmental motives. This is done through the use of a large case study which presents, documents, and discusses the theories used in the Directorate Information System project, and describes their evolution.

Part One:

Introduction, Background and Motivation

Chapter 1: Introduction

1.1 Background & motivation

1.1.1 The Initial Requirement

As a result of recent reforms of the UK National Health Service (NHS) [NHS89], [NHS90], many hospitals in the UK are adopting the clinical directorate system of organisational structure [Disken90]. The clinical directorate has a similar staff content to the old hospital department, but an additional set of more 'business' oriented roles and responsibilities. So that these are fulfilled by existing personnel a new type of information system that would support them in their activities has been proposed - a Directorate Information System [KPMG89]. Each clinical directorate would have its own system, the purpose of which would be to "support the business of the clinical directorate" [Holland92].

It was recognised that the problems associated with the creation of a generic Directorate Information System would be vast, and that similar projects had failed spectacularly (see below). However, the benefits to be gained from such a system, integrated and implemented at the level of the mair managerial unit of the hospital would be correspondingly great. To minimise the risk, and yet still get useful benefits from work that could address some of the problems outlined, a small experimental project was set up in the Diabetes and Endocrine Directorate. This project took the form of doctoral research meaning that many of the issues that have so often led to system failure could be considered in detail. Any findings of the project would be of use to the department and hospital, but by virtue of its small scale and experimental nature were the project to turn out to be useless the hospital would not suffer adverse consequences, and would not have wasted much money.

This then was the background to the study which is reported in this thesis.

1.1.2 Observed Problems and an Experimental Solution

Historically, clinical computing is an area which is characterised by expensive failures, especially when the domain to be served is an organisation (such as a department or hospital) rather than a particular function within that organisation (as would be the case with a payroll system in a personnel department or a medical record system in a cardiology department). Recent high profile problems have concerned the London Ambulance Service, West Midland Regional Health Authority's Healthtrac system, and Wessex Regional Health Authority's controversial RISP project. These systems have generally been late, expensive and have not fulfilled the needs of the users. Disasters such as these are not exclusive to the health sector - many computer projects, especially those attempting to support complex organisations, seem to suffer from similar problems. It is widely considered in the computing community that perhaps the greatest outstanding problems in the development of information systems lie in the area of requirements engineering - indeed, Fred Brooks tells us that 'no other part of the work so cripples the result if done wrong'. It is this shortfall more than any other which consistently leads to the expensive mistakes that can be seen in the health sector.

It was considered at St Thomas' Hospital, at least in the department where the project was based, that the critical breakdown in the requirements elicitation process was the communication between the users and the systems analyst responsible for the design of the 'solution'. All too often, the analyst got the 'wrong end of the stick', or misunderstood the issues being discussed. Additionally there was a feeling that the suppliers were more interested in educating the potential purchasers of their systems how the application in question was going to solve their problems than in understanding those problems in the first place.

One of the significant tasks of the project was to address these flaws in the conventional requirements elicitation process: flaws that had to be (at least partially) overcome if any progress was to be made into understanding the requirements for a Directorate Information System.

1.1.3 A Methodological Framework

The subject of the project was thus the design of an information system. One of the earliest observations made was that an information system stores, processes and transmits information about some part of 'the world'. It does this through the manipulation of symbols that are to be understood by the user as representing aspects of the world, variously referred to as the 'domain of discourse', 'domain of interest', or simply 'domain'. In other words, in use, the information system is to be interpreted into the domain. Furthermore, an information system that does not support ready interpretation will be considered unsatisfactory by its users. This is hardly controversial (although some would claim it to be over simplistic), but the identification of a problem is not the same as its solution. How can we assess an information system to see how well it can be interpreted into the domain it is intended to support? We cannot do this directly, but we can compare the properties of a computer system with properties of the user's perception of the domain in order to judge the potential for ready interpretation. In order to do this we need to have an understanding of the computer system, the domain, and the interpretation of the one into the other. The representation of the properties of a computer system is relatively straightforward, but a representation of those of the domain as it is perceived by its users is more problematic. As discussed above, there seems to be a breakdown in communications between analysts and users (clinicians): we need to address this if we are to construct a valid description of the domain.

In order to overcome the problem of a lack of understanding of the user's world view, and to avoid the necessity of long winded education of the user as to the understanding of their world gained by the analyst, the author turned to the method of empirical science as described by Sir Karl Popper [Popper80].

The scientific method allows us to test conceptual structures, known as theories, against the observed world to see if they are valid. This is done through the derivation of theorems that are purported to hold in the world, and the attempted observation of phenomena in the world that refute those theories (in which case the theory is refuted and must be re-constructed). In this way, erroneous theories can quickly be shown to be faulty and thus abandoned. That this approach is extremely powerful can be seen when the dramatic advances that have been made in our knowledge of the physical world during the last two centuries are considered. In addition, by attempting to elicit counter-examples from workers in the domain, the educative process would be of the analyst by the user rather than the other way round.

Once a good description of the domain has been elicited from the workers and other stakeholders in the system we can see to what extent a given information system might be a reasonable representation, and can strive to make improvements accordingly. This stage of the analysis is characterised by the identification of potential problems and their possible solution. The analysis process thus has two philosophically distinct phases. The first is where the nature of the domain is elicited: this (it is claimed) is based on the methods and notions associated with science. The second is where that description is used as the yardstick against which to measure the representational adequacy of an information system's design, and to change the design as a result of this measurement: this is based on the methods and notions associated with engineering.

1.1.4 Problems with Existing Methods

Early on in the project a number of techniques were investigated that might be used to conduct an analysis of the clinical directorate within a framework such as that described. It was decided that conventional data modelling techniques (see e.g. [Hull87] and [Coad91] for a summary of these) tend to focus on information rather than domain behaviour explicitly - they are good for describing the solution, but less so for understanding the problem. Soft systems analysis [Check90] appealed to us more as it seemed to concentrate on the (perceived) behaviour of the domain rather the information that would be needed to support that domain. However, the formal semantics of both conventional data modelling and the soft systems analysis notations are poor which means that any models or theories resulting from the analysis are very difficult to validate.

Not only were the formal semantics of the notations associated with methods investigated extremely poor, but the methods themselves could not be considered to support refutation in the Popperian sense. Instead of attempting to elicit counter-examples to theorems derived from the conceptual theory, validation consisted of showing the theories themselves, expressed in the method's notation, to the users and asking for their comments. This is precisely the 'user education' that it had been decided to try and avoid.

Having decided that none of the conventional systems analysis methods we had considered could be easily adapted to the philosophical stance adopted, the task of the project became more defined. Not only should the requirements for an appropriate information system be specified, but the method by which those requirements were to be found needed to be developed if the all too common mistakes of clinical systems development were to be avoided.

1.2 Objectives of Project

We can now consider the hypothesis that this project set out to test. Firstly we must again state the postulate on which the work is based.

Postulate: *That an information system in use is interpreted into a domain of activity by its users.*

The hypothesis can be summarised as follows.

Hypothesis: *That the rigorous use of a method where a semantically rich description of an information system is compared with a similarly rich scientifically derived description of the domain to be supported is possible and can prevent interpretational problems in the resulting information systems.*

This is not to say that the method alone resulted in the direct and straightforward implementation of perfect information systems - rather, when used in association with other approaches, the quality of the resulting system could be improved.

From this hypothesis, three working objectives can be derived as follows.

Objective 1: *A method such as that described in the hypothesis was to be developed.*

Objective 2: *The method should be used to derive a 'scientific' description of the clinical directorate.*

Objective 3: *From the resulting description, specifications for components of the Directorate Information System were to be engineered.*

The thesis should be read with the postulate, hypothesis and three working objectives in mind.

1.3 Structure of Thesis

The thesis is presented in four parts:

- Introduction, Background and Motivation
- Review of Methods, Method Used, and Identification of Problem Boundary
- Results: The Theories, Their Evolution, and Their Use
- Review of Method and Conclusion

1.3.1 Introduction, Background and Motivation

This part provides the organisational background to the project in a number of ways. The structural changes to the NHS which have resulted in the introduction of the Clinical Directorate into St Thomas' Hospital are explained and discussed. A brief review of the chequered history of clinical computing is then presented. Finally, some of the problems associated with requirements elicitation for information systems in general and clinical information systems in particular are described and discussed. The importance of interpretation of the information system into the domain by its users is considered. The use of the scientific method to guide elicitation of a theory of the domain is described.

1.3.2 Review of Methods and Method Used

This part of the thesis discusses a number of existing methods, introduces the hypothesis to be tested and the working objectives of the project, and describes the method used. Several domain analysis and systems analysis methods are briefly described and discussed, with particular attention paid to their shortcomings in the light of the decision to develop information systems by comparing possible designs with a scientifically derived theory of the domain. The hypothesis presented above and the working objectives are reiterated and expanded. The method that was used to create the required theories and to engineer the specifications is presented and justified by referring to the scientific method and the underlying assumption of the project - namely that successful information systems must be capable of being interpreted into the world by their users. This explanation is fairly brief - the method evolved over the course of the project, and a full discussion of the rationale, implications and limitations of the method is kept until after the results it produced are described. A brief review of a number of different models and theories of health care is presented. It is explained here that none of these was used in its entirety, but several influenced the initial design and suggested ideas for the subsequent development of the theory. Finally, the 'boundaries' of the problem are considered in the light of theories constructed by others to describe similar domains.

1.3.3 Results: The Theories, Their Evolution, and Their Use

This part describes the theories which are the practical result of the project. There are three classes of theory: the domain theory, the information system theory or specification, and the theory of the information system's interpretation into the domain, called the interaction theory. The first of these theories, the domain theory is probably the most important and the most difficult to get 'right'. Consequently the greater part of this chapter is devoted to the description of the theory, not only in its final form, but also how it evolved through the processes of 'emboldening' and refutation. Comments on

the method are made within the text as they arise. The nature of information system theories and of the interaction theories that link them to the domain theory are explained through the use of the example of the directorate's Clinical Record System (its specification and interaction theory). The process of using the three theories to engineer a (better) system specification, broken down into four 'developmental motives', is explained using the first component of the integrated Directorate Information System (being the integration of the Clinical Record System and the Outpatient Appointment System) as an example.

1.3.4 Review of Method and Conclusion

This section discusses the method in the light of the experiences gained from its use, and concludes the thesis. A more detailed consideration and justification of the method used to construct the theories and engineer the system specifications is contained here. The way this is presented is to discuss four aspects of the method - the underlying assumption, the construction of the domain theory, the use of the interaction theory, and (more briefly) the development of the information system from the specification. Each of these aspects is explored in three parts. Firstly the rationale and benefits of the method are explained. Secondly disadvantages are discussed in general philosophical terms and illustrated with cases taken from the project. Thirdly a synthesis of these two is presented suggesting the way in which we should approach this part of the method.

The conclusion reiterates the objectives and discusses the degree to which they have been met, and hence the extent to which the hypothesis has been tested. The salient features and findings of the thesis are summarised, some of the practical uses to which the work has been put described, and directions for further work expanding the findings presented in the thesis suggested.

Chapter 2: Historical Background to the NHS and St Thomas' Hospital

2.1 Introduction

There can be few people with any contact with our National Health Service who have not heard of the celebrated White Paper on health service reform, subtitled 'Working for Patients', and its associated Act of Parliament. This has been a topic of much debate and contemplation by those in authority in our hospitals and other health care institutions, and is widely recognised as the cause of a major shake up in the way that health care is provided in the UK. What is not often recognised is that the White Paper is just the latest in a long history of imposed and evolved changes in the nature of control and who exerts that control over and within the health service.

Since the NHS was originally set up in the late 1940's, its control has shifted between the patients, the electorate, the government and the doctors. Hospitals have acted as microcosms of these power shifts, with external bodies (Regional Health Authorities, Government in the form of planning committees, and District Health Authorities post White Paper), financial administrators (the hospital management and District Health Authority management pre white paper) and clinicians (mostly doctors, though recently other professions have become more influential) all having different and not necessarily complementary controlling roles which have changed continuously and extensively over the years.

As a result of its size (the largest employer in Europe[i]), diversity of stakeholder's interests, and the continuous and incremental nature of its change, the NHS is an extremely complex organisation that could take a lifetime to understand fully. Although it is not necessary to be informed as to the minutiae of the NHS and its recent structural reforms, it is important to get an understanding of the environment within which the information system to be designed will work. The complexity, confusion, dynamism and instability of the NHS in the early 1990s was an ever present backdrop to the project, and influenced our analytic stance. For this reason, a brief description of the structure of the service, and an overview of the changes and reforms to which it has been subjected are presented here.

2.2 1948 and The Beveridge Report, The Genesis of Britain's National Health Service

It is not necessary here to present a detailed description of the functioning of the service: rather this section attempts to furnish the reader with an idea of its original goals and purposes, and the managerial control mechanisms deployed to enable such goals to be attained.

One of the contentions of the founders of the modern NHS (see, for example [Webster88]) was that there was an essentially finite amount of ill health in the country, and that after an initial investment to clear the backlog of disease, the costs of the service would fall to a level that would enable any residual level of ill-health to be dealt with. It was imagined that the essence of the problem lay not in the manner in which health care was organised, but in the way in which it was dispensed to the population. If health care could be made available to all regardless of their wealth or social status, the existing hospitals and organisational structures of the health industry would be able to provide the vast improvement in the health of the nation that was envisaged. The structure of the NHS was therefore an administrative framework, dealing with reimbursement for staff, within which the medical profession could act largely as they had always done to administer care to the needy.

[i] Although following the reforms the NHS employs many of its staff indirectly through hospital trusts.

The management structure in the service was very 'loose' and fairly changeable. By the late 1970s a hierarchical management structure had been created where the Department of Health and Social Security was reported to by a number of geographically defined Regional Health Authorities (RHAs). Each RHA comprised several (again geographically based) District Health Authorities (DHAs), and each DHA had direct control of all acute and community hospitals that lay within its boundaries. DHAs were governed by a board of permanent officers which consisted of a chief administrator, a senior nurse, a community physician and representatives of hospital doctors. Decisions at board level entailed consensus (which effectively meant unanimous) agreement amongst members of the board. The large size of DHA boards and the requirement for unanimous decision making meant that hospital management was ineffectual, and funding decisions usually resulted only in small adjustments to previous budgets [Harrison92].

If the initial assumption about the nature of ill health in the country had been correct, then the original organisational structures of the service may have sufficed to deliver the benefits promised. It would seem however that the demand for health care, if not infinite, certainly outstrips any reasonable level of supply, and will always do so. Eradication of common illnesses simply means that people will be well enough to suffer from previously less common illnesses. We can see that this is the case with the most common fatal diseases changing from what had historically been the biggest killers - tuberculosis, smallpox, polio - to new conditions that are more expensive to treat - heart disease, and cancer [NAHAT91]. Not only will people always die of something, and want to be cured of that, but the healthier they are, the healthier they expect to be. We have largely succeeded in eradicating polio in this country: the population is not satisfied with this advance and now expects to be treated for cancer and heart disease. This increase in expectations for health status is reinforced by the availability of many new and costly forms of treatment: hearts can now be transplanted, cancers (in many cases) cured, brain tumours excised and so on. Not only does the public expect better health care, but that care is potentially deliverable - at a price.

As a result of this progress the costs of health care in the developed world have risen enormously over the past few decades. In fact, the percentage of the UK's Gross Domestic Product spent on health care has risen from 3.9% in 1960 to 5.9% in 1984 [Schieber87]. The problem is as great if not worse in other developed countries: the mean health spending as a percentage of GDP of all OECD (the Organisation for Economic Co-operation and Development) countries has risen from 4.2% to 7.5% over the same period (the United States spends 15% of its GDP on health care available to only 63% of the population). The many reforms of the NHS since its inception have almost all been focused on the need to cut the huge costs of health care through the improvement of its management.

2.3 Loss of Innocence: The Griffiths Reports

1979 saw the election of the most radical national government for many years. Margaret Thatcher and her ministers had a profound agenda for change that was felt in all areas of national life. The NHS was no exception to this, and the Conservative administration ushered in a succession of sweeping structural changes to its control and management between 1982 and 1989. The first major review and subsequent re-organisation was instigated in 1983 by the then Secretary of State for Health and Social Security: Norman Fowler. The review was

'to examine the way in which resources are used and controlled inside the health service, so as to secure the best value for money and the best possible services for the patient [and] to identify what further management issues need pursuing for these important purposes' [NHS83].

The review was headed by Mr Roy Griffiths, the managing director of the supermarket chain J Sainsbury who published his team's findings as a letter to the Secretary of State in 1983 [IHSA83].

Griffiths' assessment of the problems facing the health service centred on the weakness of management control and its lack of vision. In the report the inquiry team claimed:

'...it appears that consensus management can lead to 'lowest common denominator decisions' and to long delays in the management process ... In short, if Florence Nightingale were carrying her lamp through the corridors of the NHS today, she would almost certainly be searching for the people in charge' [*ibid* pp 17 & 22]

and

'...there is no driving force seeking and accepting direct and personal responsibility for developing management plans, securing their implementation and monitoring actual achievement' [*ibid* pp 12]

The proposed solution to this lack of effective management was to set up a new and more direct management structure which became known as 'general management'. Instead of each layer of the NHS hierarchy (Department of Health and Social Security, Regional Health Authorities, District Health Authorities, and hospitals and other units) being managed by a committee which acted through consensus, there would be a general manager at each level who would be responsible for achieving that level's objectives. In order to overcome the inevitable medical resistance to this *de facto* shift in power and authority to non-clinical administrators, doctors were to be encouraged to take a more active role in the management of their hospitals. The way in which this was to be achieved was through the introduction of 'management budgets' for consultants which would act as a guide (rather than a strict control) within which they could plan and against which they could monitor their workload.

The recommendations of the review were implemented by the Secretary of State in June 1984. Although the final management structure was not exactly as had been suggested by Mr Griffiths, general managers were appointed at Regional, District and Hospital level, and management budgets were introduced with a limited degree of success in a number of hospitals.

Much has been written about the impact of the so called Griffiths report on the health service whose detail need not concern us here. We should note that the effects of the report are considered to have been profound, but partial [Politt91] in so far as they have re-defined the structure of the NHS's organisation.

2.4 Rapprochement and Reconciliation: The Resource Management Initiative and the Clinical Directorate

The introduction of management budgeting was envisaged as bringing doctors and managers closer together, and to enable medical considerations to be fully represented in resource management decisions. Three years after its inception, a joint statement issued by the Joint Consultants' Committee and the NHS Management Board [NHS86] roundly condemned the slow progress made by that part of the recommendations of the Griffiths Report. They decided that

'... many doctors ... have still to be convinced that management budgeting is more than an accounting exercise which simply increases overheads for no commensurate benefit.' (*ibid*)

The problem was the continued perceived lack of relevance of information available to individual service providers - the information was too crude for a clinician to use to benefit his or her practice. The remedy to the problem was a new project that became known as the Resource Management Initiative. This was initially an experimental project based at six English hospitals and then introduced more broadly to hospitals throughout the country (the initial hospitals were the Royal Hampshire County Hospital,

Freeman Hospital, Guy's Hospital, The Royal Infirmary Huddersfield, Arrowe Park Hospital and Pilgrim Hospital).

The main objective of the initiative was to see

'... how doctors and nurses can be involved such that they are committed to the management process, responsible for their use of resources and able to take better decisions regarding service quality and service care'. (*ibid*)

The initiative was evolutionary in nature, and as Buxton et al assert

'... it has developed more in reaction to previous problems with national and local initiatives than in fulfilment of a clearly specified model' [Buxton89].

However, by the end of the third year of the experiment, certain patterns had emerged. One of these was in new structures of hospital management, notably the Clinical Directorate Model.

Although the Griffiths report changed the overall organisational structure of the NHS, the way in which doctors and managers liaised remained fairly unchanged (for a discussion of this see [Harrison92]). Due to the failure of Management Budgeting, the proximity of control of operating budgets remained in the hands of the hospital managers. Doctors reported to the hospital board on medical matters via the Medical Executive Committee - a representative body of consultants. Nurses were similarly represented, as were other paramedics. This traditional structure is illustrated below:

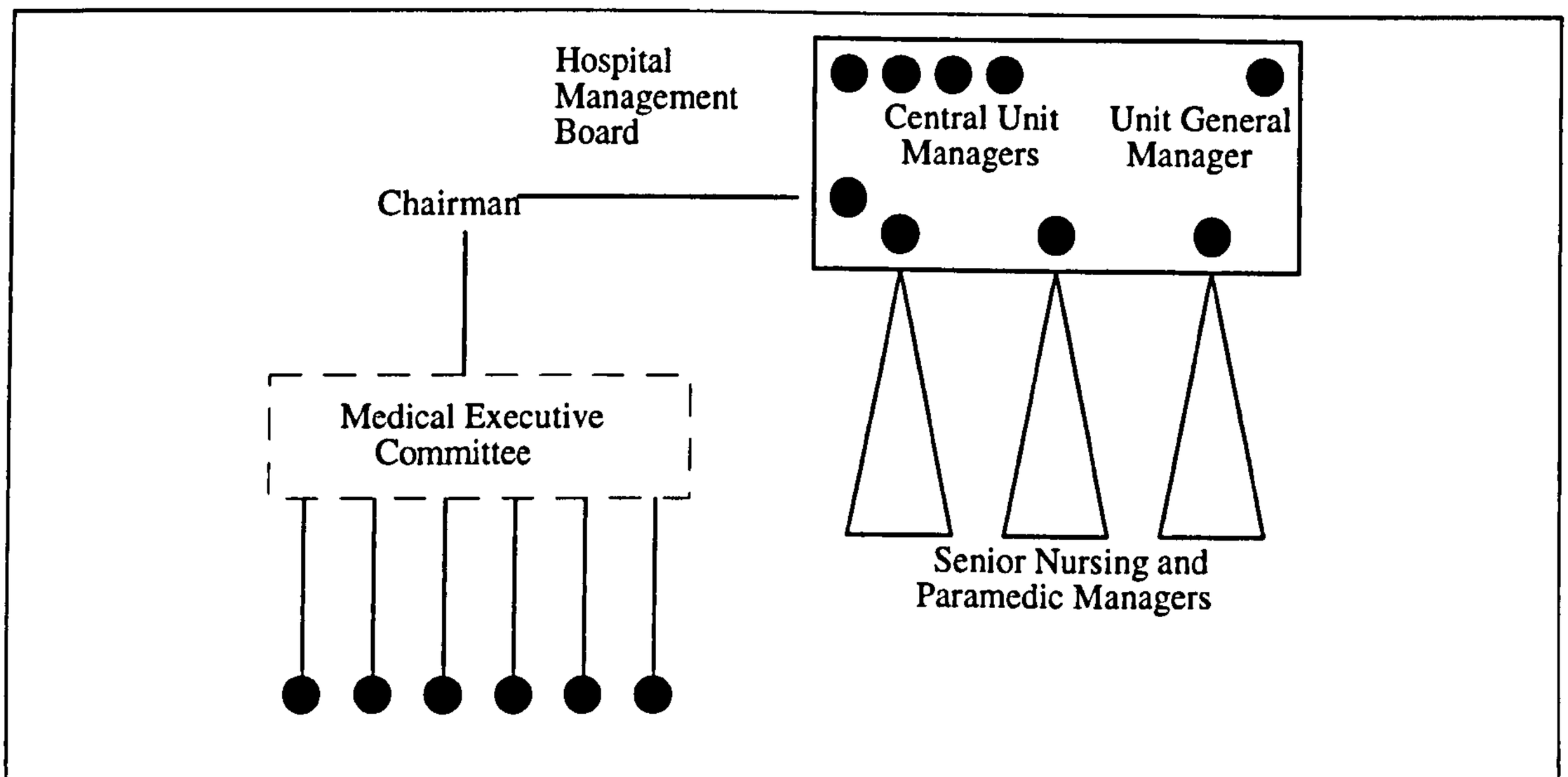


Figure 1-1: Traditional Model of Service Providers' Participation in Hospital Management (from [Buxton89])

The clinical directorate structure of hospital management represented a clear break from this conventional model. In the new system, each doctor, consultant, nurse and paramedic belongs to a specific Clinical Directorate. A Clinical Directorate's boundary would be defined such that the tasks carried out within it were coherent in a medical sense. Thus a hospital might have an Paediatric Directorate, a Renal Directorate, a Surgical Directorate and others. Each directorate was to be managed and led by a Clinical Director (normally a consultant, though occasionally a nurse) who might be assisted in the day to day aspects of management by a business manager attached to the directorate. The directorate would in turn be represented at hospital board level by the director. The Clinical Director acts as prime budget holder

for the directorate, and often has the ability to increase the directorate's income or change the proportion of resources allocated to different sub-divisions at least semi-independently of the hospital as a corporate unit.

A schematic from the same source as the previous one is given below which illustrates this concept.

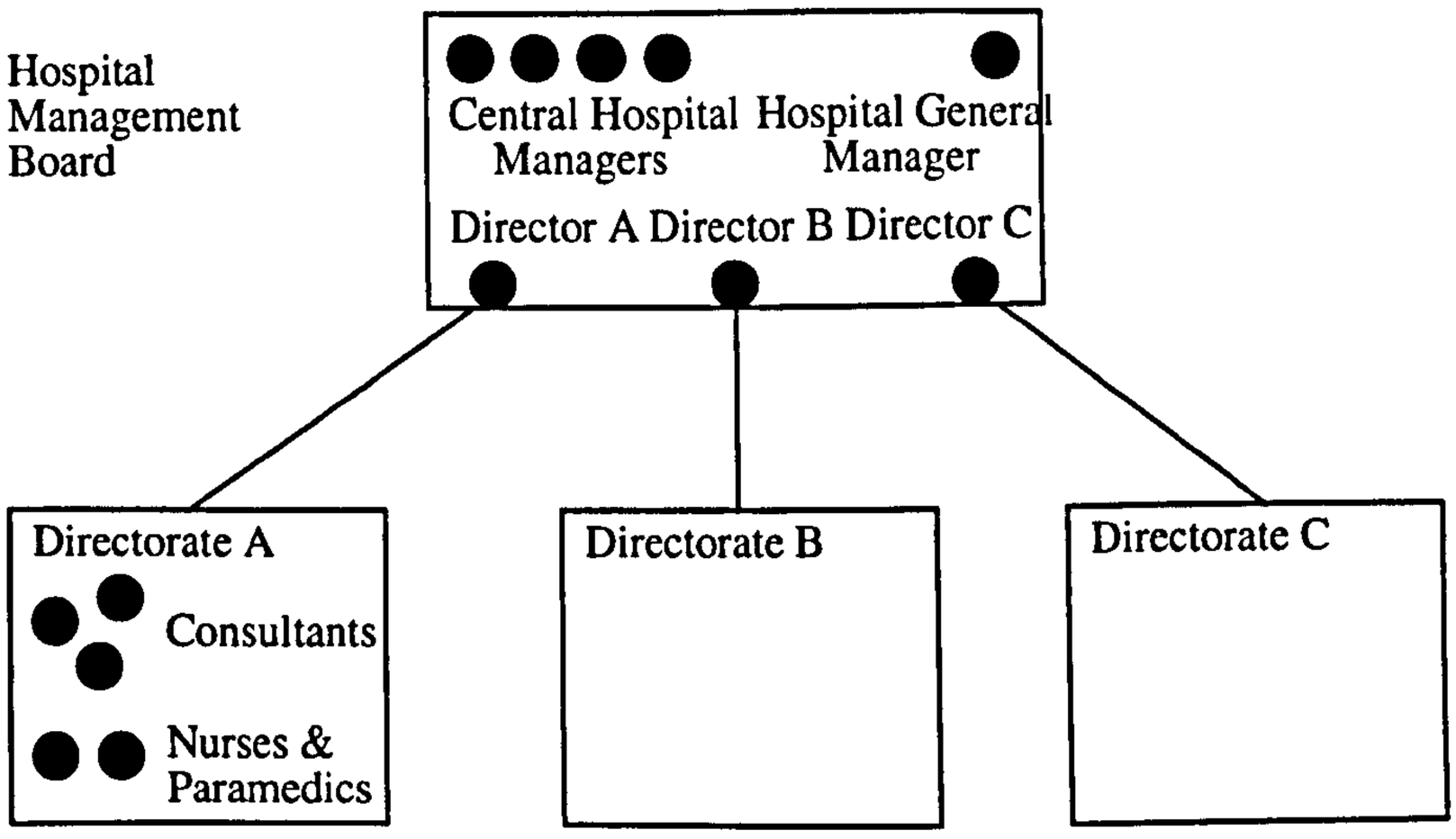


Figure 1-2: Clinical Directorate Model of Service Providers' Participation in Management (From [Buxton89])

The progress of the Resource Management Initiative was to be reviewed and criticised by a team of researchers from Brunel University. This team published their findings in 1991 [Pack91], by which time the government had already decided that the experiment had been successful, and had encouraged take-up of the main ideas by other hospitals. The team concluded that the new Clinical Directorate structure constituted a fundamental change to the process of hospital management, in particular representing

'Two transformations: from an organisation based on a division of labour to one based on a division of knowledge; and from top-down to distributed control' [*ibid* pp164]

One might liken the hospital to a large holding company in which case the clinical directorates would be operating companies within it. The clinical director could be compared to the chairman of the board of a holding company and the business manager to the managing director.

Few hospitals have implemented the Clinical Directorate system exactly as described above [Disken90]. For example the nursing staff might be represented on the hospital board *en masse* as well as through the directorate structure, or the Clinical Directorates might not be directly represented at board level, but would be grouped together into aggregate units. This is the case at St Thomas' Hospital which can be regarded as having a fairly typical Clinical Directorate structure for a large teaching hospital: it has thirty-five Clinical Directorates which are aggregated into six Group Directorates, each of which is represented on the hospital's management board.

Although the Clinical Directorate structure of hospital management has been widely copied, it has not always brought the benefits expected of it. In many cases the autonomy of directorates has been fairly limited, and although the technical lines of responsibility and authority have changed, the way day to day

control is exercised has remained basically unchanged from how it was before the Resource Management Initiative was 'rolled out' across the country. This is partly because the information systems to support this distributed fashion of management do not for the most part yet exist, meaning that management information for clinical directors is still 'controlled' centrally (by the hospital management, usually the finance department), rather than 'locally' (by the clinical directorate). Clearly this point is central to the thrust of the PhD and will be discussed at length later in the thesis. Not only has the information not always been available in the appropriate form to support the new structures, but in many hospitals there is such a pervasive and continuous lack of funds that there is effectively no 'room to manoeuvre' on the part of the directorate, and all that the clinical director can do is make the extremely minor changes in organisation and process that are affordable.

A last point is that one of the aims of the RMI was to bring the clinicians and managers in the country's hospitals closer together, through involving clinicians more effectively with the management process, and giving them managerial responsibilities. This has been partially successful through the use of clinical directors as clinician advocates and having them sit on the hospital board, however there is the danger that those clinicians that elect to do so are considered 'management stooges' by hostile fellow professionals. Critics of the system also point out that while it might seem that clinicians, through their clinical directors, now have a significantly more powerful voice at hospital board level, this is not always so; before, they spoke as one voice through the chairman of their medical executive committee: now, they have a number of different representatives that may very well not be in agreement with each other, especially when it is a case of different services competing for a share of the hospital's budgets - in short a policy of 'divide and rule' is enabled which actually amounts to a diminution of the power of clinicians when compared with central hospital management.

2.5 Recent Changes: The White Paper and the Tomlinson Report

Three years after the Resource management initiative was launched, and before a considered assessment of its affects was ascertained, the government decided to launch the most radical and far reaching reform of the NHS to date. The changes were heralded in the 1989 White Paper: Working for Patients [NHS89], which was enacted as law in 1990 [NHS90].

The central thrust of the new reforms was to create an 'internal market' for health care within the NHS. The mechanism for achieving this has been the division of the service into 'provider' and 'purchaser' units. Providers are generally hospitals and other locations where direct care is proffered, whereas the purchasers acted as the customers of the services provided, and could choose from available providers which it would patronise with its custom. In this way it was hoped, a degree of competition would be introduced into secondary care: a health-care market would emerge where money was spent and resources committed according to the needs of the consumers (as represented by the purchasers) rather than the whims of the providers or simply historical precedence.

To underline the separation of function between the purchaser and provider units, the government enabled and encouraged hospitals to remove themselves from DHA control and set themselves up as semi-independent 'trusts'. Although still part of the NHS, these trusts would be able to sell their services to the highest bidder (subject to certain constraints issued by the Department of Health), and could re-invest any profit they made into hospital services. The trusts could borrow money on a commercial footing, and could negotiate local pay and conditions for their staff. At the time of writing most of the UK's NHS hospitals are trusts, and the Secretary of State has indicated her desire to have all hospitals achieve this status in the near future.

There are two models of the health market that exist side by side in the current NHS. The first, which is currently the most common, is for the DHA to act as an advocate for its local population, assessing health needs of that population and buying services from possible providers so as to satisfy those needs in the most efficient manner possible. In this model the role of purchaser is split between the health authority and the GPs who tend to their patients. Whereas the health authority is responsible for negotiating and securing contracts for services from provider units, GPs are the people who use those services. The GP is not an employee of the DHA, and in fact is not technically an employee of the NHS at all but is in fact self-employed (this anomaly is described clearly and succinctly in Abel-Smith [Abel92]). The GP can technically refer a patient to any hospital he or she feels would provide the best care for the patient, but the hospital is likely to only accept those patients who are funded, either through contracts with their DHAs or through an unwieldy mechanism known as Extra-Contractual Referring. GPs themselves are funded through their local Family Health Service Authorities (FHSAs) which are generally different in size and boundary from any DHAs.

The second model is simpler and has the GP as purchaser, acting on behalf of his or her patients to secure the necessary services through 'shopping around' provider units. This model only applies to a category of GP practices known as GP Fundholders (GPFHs). In order to be eligible as a GPFH, the practice must have more than 9000 people registered with it: the number of people so registered is known as the 'list size' of the practice. GPFHs are allocated their own budgets according to a formula, based on their list size, known as 'weighted capitation'. A GP who works in a fundholding practice can refer a patient to whichever hospital he or she chooses, but must pay for the service delivered out of the practice's budget.

The changes outlined above are sweeping and profound, and have caused turmoil within the service. Some of the problems have been caused by the internal inconsistencies of the white paper on its own and when combined with previous un-retracted initiatives, and others through the brutality of any market system when allowed to operate without any centralised planning and control. An example of the internal inconsistencies within the new NHS can be observed in the existence of two different models of the health market that exist side by side. This has been noted elsewhere, and Abel-Smith considers that

'It is curious that the government decided to go for both approaches all over the country at the same time: districts as buyers of services *and* fundholding practices as buyers of services. Presumably this was a compromise between competing views on the best way ahead'.

Critics of the reforms have claimed that the existence of these two parallel systems has led to a 'two-tier' health service where patients of fundholding GPs get preferential treatment over patients of GPs that are not fundholders (Although this is generally denied by ministers there are persistent accusations that 'queue jumping' occurs in most hospitals, and is practised by many of consultants [Water93]).

The potential brutality of the health market is a worry to the Government. The place where this might be most keenly felt is London where it is considered that there are too many hospitals very close to each other, all providing similar services to a population that has been dwindling steadily over the past hundred years. It was felt that if left to the market to cull this surfeit of hospitals, years of transitional chaos might result where individual departments were gradually picked off, leaving all hospitals diminished and capable of providing only a low quality of service. To forestall this the Secretary of State announced a review into the provision of healthcare in London, headed by Sir Bernard Tomlinson. He delivered his report entitled 'Making London Better' on the 23rd of October, 1992 [Tomlin92] amidst intense media speculation. The report stated that the disintegration of the provision of health care in London could only be prevented through planned closures and mergers of many of the most famous

hospitals in Britain. The hospitals themselves have fought back with high profile media campaigns and criticisms of the data the review team used to come to its conclusions. At the time of writing it is still not totally clear how secondary health care in London will change over the next decade: for example, although Guy's and St Thomas' hospitals have been combined to form a single trust (the Guy's and St Thomas' Hospital Trust), it has still not been decided whether one of the sites should close entirely, and if so which it should be. The result of this procrastination is that the market has started to wreak the havoc predicted, with University College Hospital and Middlesex Hospital being unable to carry out non-emergency work for their major purchaser, Islington and Hackney DHAs. This in turn has lead to strikes and accusations that the Government is turning the NHS into an emergency only service.

2.6 Conclusion: Complexity and Confusion

This chapter has been a brief introduction to the organisation of and the recent changes in the UK's National Health Service, as seen from the vantage point of a major London Teaching Hospital. Much has been skated over such as the nature of community services which have been gaining prominence in recent years (One of the major trends on health systems in all developed countries has been a steady shift in importance from acute illnesses that have been served by hospitals to chronic illnesses better served in the community [Jones91]). The PhD not only used St Thomas' Hospital as the subject of its research, but was based there with the author having an office in a clinical area of the hospital (as opposed to a managerial or administrative area). As a result the daily issues that were considered important to the staff of the department where the project was based were also considered important to the author, and have influenced the analysis which has been the topic of the work. For this reason a basic grasp of the organisational and political environment within which the hospital exists is important if the reader is to understand some of the decisions that were taken as part of the analytical process.

The most important impression that the reader should be left with is one of extreme organisational turmoil that has recently engulfed the health service and the confusion and organisational complexity that has resulted. We should note that none of the changes and reforms described above has resulted in a wholesale redefinition of the existing structures and definitions. Each change has been incremental and has added to the previous system. The current NHS has aspects of function and structure that can be traced back to any of the organisational changes we have looked at. That the different reforms represent extremely different (and non-complementary) political and managerial philosophies has merely added to what has always been a culturally and politically complex environment. When we consider additionally the recent turbulence over the Tomlinson report and the destruction being wreaked on the historically stable London Teaching hospitals by the mechanics of the 'internal market' we should not be surprised to discover that the prevailing culture of the NHS in London is one of bewilderment and confusion tinged with frustration and despair.

In short, we leave this chapter cognisant of the facts that the NHS has been subjected to a series of extreme and rapid changes, and that the combination of these has left the employees of the service confused and unclear as to the effects that these changes will have on their work.

In the next chapter we will see how the changes described above affected St Thomas' Hospital, and how they resulted in the expressed need for a new type of computer system - a Directorate Information System.

Chapter 3: St Thomas' Hospital and the Directorate Information System Project

3.1 Introduction

In this chapter, we will investigate the background and rationale of the project that is the subject of this thesis. St Thomas' Hospital is described, and its decision to introduce the specific clinical directorate structure as a response to the challenges it faced at the end of the last decade is explained. Computing in the NHS has historically concentrated on management and strategic systems rather than operational ones. This anomaly is explained and criticised in two of its guises - at the level of national policy in the form of the case-mix management specification and locally in the form of the hospital's information technology strategy. One way of addressing this anomaly was considered by St Thomas' - the integration of central and local systems at the level of the directorate to create 'Directorate Information Systems'. The concept of the Directorate Information System, and the nature of a project that would investigate the idea in some depth is explained here. Finally, some functions of such a system are presented in order to give substance to the concept so as to help the reader in the following chapters.

3.2 St Thomas' Hospital and the Introduction of the Clinical Directorates

3.2.1 St Thomas' Hospital: An Ancient Institution

St Thomas' is one of the oldest, largest and most famous hospitals in the country. Founded by Augustinian monks in 1106, it moved to its present site in 1871. The hospital has over 600 beds, and treats 45,000 people as inpatients every year. Each year, 254,000 appointments are made for the outpatient clinics and 70,000 patients are seen at the Accident and Emergency department. In addition to the large amount of routine medical activity that takes place in the hospital, as a teaching institution (the associated medical school is incorporated into the University of London) much effort is expended on educating student doctors and nurses, and on research into a multitude of medical conditions.

The hospital has tended not to exploit the organisational changes in the health service as immediately as others: for example, St Thomas' was one of the last hospitals in London to be granted trust status. It has always been willing to learn from the experiences of others however - the introduction of clinical directorates is no exception.

In the late eighties, the hospital suffered from poor financial management, and an associated string of scandals. It was badly in debt at a time when the government was increasingly insisting on 'balanced books' in the public sector. Poor financial performance associated with rumblings about imminent shake-ups in London's healthcare led to low staff morale and antagonism between clinicians and administrators. Guy's Hospital, two miles east of St Thomas', had suffered from similar problems for many years. In 1985 it introduced the clinical directorate management structure, and subsequently became one of the first hospitals to take part in the Resource Management Initiative. Since this time, it was generally considered that the management of the hospital had improved markedly - it was no longer so indebted, and the morale of the staff was dramatically increased. The management at St Thomas' decided a system similar to that at Guy's hospital could be usefully set up there. Consequently St Thomas' applied to join the resource management initiative in 1988 and implemented its own Clinical Directorate management structure in 1989.

3.2.2 The Directorate Structure at St Thomas'

The structure that was chosen for St Thomas' was slightly different from that at Guy's. It was considered that the prevailing culture at the former hospital would accommodate many small tightly defined

directorates rather than a few large ones that covered a number of specialties. Whereas Guy's introduced 6 directorates, St Thomas' set up over 30. The size of a clinical directorate was thus similar to that of a department of the medical school. It was clearly infeasible to have 30 clinical directors sitting on the hospital board, so the directorates were grouped into a smaller number of larger administrative units: the group clinical directorates. There are six group clinical directorates at St Thomas - the Diabetes and Endocrine directorate is in the general medical group: group two. Each group directorate is headed by a group director, who sits on the hospital's management board, representing the interests of the directorates he or she leads.

This structure seems to have been generally accepted at St Thomas', and is regarded as one of the factors that has changed the hospital from being the lame duck of London hospital healthcare to one of its most successful components: the hospital is one of the few that actually faired relatively well out of the recent review carried out by Professor Bernard Tomlinson (St Thomas' is to merge with Guy's, but most of the clinical services will be provided on the St Thomas' site). In fact, the entire Resource Management Initiative is generally considered to have resulted in better, more responsive hospitals, and improved patient care by a wide variety of clinicians. This is in marked contrast to the hostility with which the changes associated with the 'Working for Patients' white paper are viewed.

3.3 Local and Operational Information Systems: The Weak Link in NHS Information Technology

3.3.1 The Case-mix Management System and its Feeders

The introduction of the clinical directorate organisational system was only one part of the Resource Management Initiative. Equally important to the managerial structure were to be the information systems that supported that structure. The clinical directors needed to know how their directorate was performing in relation to the various activity, financial and quality targets that were agreed at board level. The information needed would be of an unprecedented sophistication, and a new type of computer system was specified: the case-mix management system [Bullas89]. This would enable clinical directors (and others) to investigate how the directorate was performing in all the specialties and sub-specialties it was involved with. Individual cases could be presented, or aggregated into clinically meaningful categories, called Diagnosis Related Groups (DRGs). What proportion of the case load was caused by which DRG, in other words the case-mix, would be one statistic pertinent to the management of the directorate that could be generated by the system. There were many others besides such as waiting times, length of stay statistics, activity rates of different clinicians and so on.

It was not intended that the case-mix management system would be an operational system that data would be directly entered into - it was really a sophisticated database reporter that could manipulate existing data so it would be useful to those running the hospital. The operational systems that provided this data were called the 'feeder' systems. The purpose of these systems and their specifications were not provided other than a description of the data they needed to supply to the case-mix management system. Herein lies the major problem with the Resource Management Initiative (RMI) - the feeder systems generally do not exist, and where they do, the information is often in a form which is incompatible with the case-mix database. As a consequence of this, most hospitals employ data-entry clerks to enter data from the patient's notes directly into the case-mix system - this is expensive and extremely inaccurate. Doctor's notes are notoriously illegible, and the clerks tend to be poorly trained and motivated.

It was clear that more needed to be done at the feeder system level. This is the germ from which the notion of the Directorate Information System sprang.

3.3.2 IT Strategy Document - Practice management systems

St Thomas' Hospital has been heavily involved in the leading edge of IT in support of Health Care, and for many years was the most technologically advanced hospital in the country in this area. More recently, IT stagnated somewhat in the hospital, with little investment and an erosion of energy and morale in the computer department. The hospital decided it wanted to regain its pre-eminent position in medical computing and commissioned a study by the management consultancy KPMG to this end. The study took several months to complete, and was delivered in December 1989. The initial report [KPMG89] was not enthusiastically welcomed by either clinicians or the computing department. It was re-worked by the St Thomas' computing department, and this new document has formed the basis of the hospital's IT strategy, a developmental framework funded for 6 years by the St Thomas' special trustees.

At the time of writing of the IT strategy document, a great deal of time and effort had already been spent addressing the information needs of acute hospitals. Most of this was not directed at discrete task related systems, but rather at single monolithic systems serving entire hospitals. Many millions of pounds have been spent on the development and implementation of these Hospital Information Support Systems, or HISSs. Many workers feel that the achievements of the HISS projects are not commensurate with the time and effort invested however, and that more capable systems are needed if the goals of the RMI, and of hospitals in general are to be met. Indeed, all of the three pilot HISS sites had the systems delivered late and millions of pounds over budget. Indeed one, at the Nottingham City Hospital, has still to be completed, more than four years after the HISS initiative started.

There are two conceptual reasons why the efforts expended on the HISS projects may be misguided. Firstly there is a conflict between the distribution of control and management represented by the Clinical Directorate structure, and the centralisation of information definition and provision represented by the HIS systems. A hospital which uses a monolithic HISS and has the Clinical Directorate structure will have to make an unhappy compromise between the conflicting philosophies of corporate compartmentalisation with semi-autonomous business units on the one hand, and of central co-ordination and control on the other. The second reason, allied to the first, is that a hospital is too complex an organisation to be analysed, investigated and supported 'in one go'. The amount of work required to devise a system that supports the business of a hospital with several thousand employees and such an enormous multiplicity of goals and functions is vast. None of the HISS sites that has opted for the centralised approach has a system which could be said to support more than a subset of the hospital's business requirements, and the majority have spent many millions to achieve this (see for example [NScApr92]).

The hospital decided that the monolithic 'big-bang' approach to hospital information systems was misguided, and opted to go for a number of smaller individually less ambitious systems that satisfied clearly defined needs which could be integrated together to form a larger more complex system which would approximate the desired function of the HISSs.

In addition to this architectural decision, there were several guiding principles to the development of the strategy, some of which are spelt out below:

- Data should be collected at its source

In the past data to support the management and planning functions of the hospital was gleaned from discharge summaries, patient records and GP letters, and entered into the hospital's computer system by 'coding clerks'. It was considered that this was inefficient - partly because of the necessity of employing large numbers of clerks to carry out the uncreative data transfer activity, and partly because the data

transferred tended to be very inaccurate. It was considered that if data could be entered by the person seeing the patient, at the time of the encounter (as opposed to retrospectively in batches), then the quality of the data would be improved, and the coding clerks could be re-deployed.

- Data should be collected once only

Hospitals are notorious for demanding the same information of the patient many times. Part of this is necessary - it is best for each doctor who sees the patient to hear about his or her condition 'from the horse's mouth'. Other information should only be collected once: the patient's demographic details, details of their family and GP, and much medical information do not change, or do not need to be explained to each health care professional who comes in contact with the patient by the patient themselves. However, often these details are collected many times - once for each information system, be that a paper based or automated one. This is not only wasteful in terms of time taken to collect and record the information being sought, but can also lead to discrepancies between the different systems resulting in ambiguity and consequent degradation of data quality and reliability. By integrating together the information systems (a process which, while difficult for paper based systems is at least technically straigh.forward for automated information systems), this duplication of data entry can be prevented.

- 'Management' information should be a by-product of operational care.

One source of hostility between clinicians and administrators has been the necessity for the former to collect data that appeared unimportant in order to satisfy the seemingly arbitrary demands of the latter. This again led to time being diverted from patient care to form-filling. The idea behind this guideline was that clinicians and other 'operational' staff should only collect data that they thought relevant. Through appropriate system integration and data reporting, the information that managers needed to manage would be provided automatically. The idea that management information can be a by-product of the operational aspects of the hospital's business is essentially an article of faith - it is not obvious that it is valid, and has yet to be tested in practice.

These guidelines contain implications about the makeup of the hospital's information systems. One of these implications is that systems must be in place in order for the operational staff to record useful aspects of their work, to aid them in their jobs, and to provide the information needed by managers. The original IT strategy document described a number of these - appointment systems, bed management systems, pathology systems and nursing systems. It also recognised that much work is carried out by individual clinical practices that would not be covered by any of the above systems. These clinical practices (or departments) would be provided with so-called practice management systems to help them run their activities. These too would act as feeders to the rest of the hospital's information infrastructure. However, as with the Case-Mix feeder systems, the practice management systems were not specified, nor was their purpose described. Thus again, much of the logic of the information system plan depended on a class of systems that neither existed nor was specified anywhere.

3.3.3 Desire for Directorate Autonomy

There is a long established tradition of distinct groups of people working with a great deal of autonomy in the hospital. This is not unique to St Thomas': the aim of most junior doctors is to become either a General Practitioner, or, if they stay working in hospitals, a consultant. At a large hospital like St Thomas, there will at any one time be around one hundred consultants on the staff. Although once a consultant you can gain in experience, respect and reward, as far as the official medical hierarchy is concerned, this rank is the highest achievable. Thus no consultant is medically responsible to any other,

and each enjoys almost total discretion when it comes to planning and delivering medical care. Not only are consultants essentially independent of one another, but the entire medical profession has fought successfully to keep responsibility for the major component of hospital care - the medical process - to itself. Thus doctors are autonomous *en masse* with respect to the hospital management and (once they become consultants) they are autonomous individually with respect to each other. This tradition of independence is one of the reasons for the popularity of the clinical directorate structure, which also encourages autonomy, albeit to a slightly larger group than consultants' firms.

Now, we have seen that both the Resource Management Initiative and the hospital's IT strategy hinge on systems being run at an operational level - respectively feeder and practice management systems. The functions that these operational systems might provide could be aggregated to form discrete systems in a number of ways. An obvious way of grouping functions (and one that has been used in the past) would be hospital-wide by task. Thus we might have a bed-management system, a nursing management and information system, an appointment and resource allocation system and so on. However, implementation of these centrally based systems would not reflect the enthusiasm for autonomy in the clinical directorates. Clearly some way of consolidating the economies associated with central information systems and the increasing independence of clinical directorates was needed. It was envisaged that this consolidation would be achieved through the introduction of Clinical Directorate Information Systems.

So we see that the IT strategy adopted by St Thomas' insists not only on the distribution of computer functions operationally, but also local identity for those systems. In other words, they must be integrated at the level of the directorate.

3.4 The Concept of a Clinical Directorate Information System

3.4.1 The Original Idea

The idea of the Clinical Directorate Information System (which shall be referred to as simply the Directorate Information System, abbreviated to DIS, below) was derived from that of the practice management systems proposed in the original IT strategy document delivered by KPMG, but with the ill-defined 'practice' redefined as the clinical directorate.

It was envisaged that the Directorate Information System would not be so much a discrete identifiable system, as an integration of hospital-wide systems with local systems in such a way as to provide useful functions to the directorate. It was hoped that through the integration of the different subsystems available in the hospital, each class of user would be able to use the particular selection of such subsystems appropriate to its tasks as a single computer system. The user would not (necessarily) be aware that the particular functions she was using were implemented as a collection of separate subsystems from a diversity of suppliers passing data between different databases according to standard protocols. This perceived or 'virtual' system would be different for different users exploiting different functions, but it was expected that there would be different classes of users. One of the most important class of users, or user 'constituency', would be the operational clinical staff - doctors, nurses, paramedics, clinic clerks and so on: the people that go to make up clinical directorates. The virtual systems used by different directorates might well be different but would have common features: perhaps at some sufficiently abstract level they might be considered to be of identical form. The proposed common virtual system for clinical directorates, tying together 'real' hospital-wide and (similar) local systems is what was called the Directorate Information System.

3.4.2 The Directorate Information System Project

To investigate the Directorate Information System further, two parallel approaches were taken. The first was to react to the most pressing needs of directorates in an *ad hoc* manner, according to the requirements of the directorates that seemed to be most urgent. As a result of this approach many directorates can now access copies of centrally held word-processor, spreadsheet and electronic mail programs. In addition, several hospital wide systems such as the patient registration system and patient booking system (for inpatients) can be accessed from terminals based in the clinical directorates, although these have not been integrated at the directorate level.

The second approach was to set up a small experimental project looking at the requirements for such a virtual system in a more structured and formal way. At the end of the project it was hoped that we would have an abstract design for a DIS which could be constructed out of existing systems and new ones as required. Necessarily the analysis and design for this system would be highly abstract, avoiding as far as possible technical issues relating to the particular implementation of subsystems chosen for any particular function.

In having two approaches to the problem, the hospital risked little and could potentially gain much. If the formal project failed to produce any requirements or ideas, then at least the directorates would be sure to derive at least some tangible benefit from the *ad hoc* approach, and not much money would have been wasted as the experimental project was small in scope. If the formal project provided valuable insights into the problems associated with integrated information systems integrated at the directorate level, then the benefits to the hospital could be enormous.

The formal project was based in the Diabetes and Endocrine Directorate: the initial work would concentrate on this directorate, with the lessons learned being tested in other specialties in due course to see which were specific and which were common to all the clinical directorates in the hospital.

The progress and findings of the experimental project constitutes the main subject matter of this thesis.

3.4.3 Information Systems in Directorates or Directorate Information Systems

We should note here that all directorates have a directorate information system of sorts already as they all do perform the functions required of them. Information to support patient care is provided by the paper case notes and the availability of test results. Patient administration is supported through the existence of clinic lists, hospital waiting lists, a variety of call / recall systems and so on. Directorate management is supported through the provision of simple financial reports from hospital management, a profusion of medical audit systems, and informal quality monitoring by members of staff.

The difference between a Directorate Information System as envisaged in the IT strategy and information systems that are used in directorates is that these latter are not 'coherent' or 'integrated'. The information system components currently used in directorates do not support each other or reflect the fact that each activity should be viewed not so much as a separate self-contained function, but rather as a part of a large and complex process - the running of the Clinical Directorate. A Directorate Information System should tie these functions together into a robust whole such that each activity will be able to support the others.

3.4.4 Some Functions of a Directorate Information System

In order to give a better understanding of the sort of thing a Directorate Information System might be, some specific function areas that would be catered for by such a thing are listed below.

The administrative functions of the system would support those processes that ensure that the right people are in the right places at the right times, and that they take part in the right activities. These include referral handling, patient acceptance, clinic booking, patient admission, and patient discharge[ii].

The clinical functions of the system would support the specifically clinical aspects of the directorate's operational activities. This means the provision of information relevant to the clinical care of a particular patient and includes the case note system, test results reporting, and any advisory / care guidance / care control system.

The managerial functions of the system would provide information to enable the monitoring and change of the various activities of the directorate so that they more appropriately provide its 'goals'. These include support for medical audit (clinical management), resource management, non-clinical quality management, and higher order management such as market analysis and management. In addition these components of the system would deal with financial monitoring and reporting, a increasingly important part of the directorate manager's responsibilities.

As pointed out previously, all these functions are already supported in one way or another. The Directorate Information System will tie them all together in such a way that they support the 'business' of the Clinical Directorate.

3.5 Conclusion

We saw in this chapter that one of the missing components in integrated clinical computing both nationally and at St Thomas' is a type of information system that can work locally supporting the operational divisions of the hospital. Where local systems do exist, they do not reflect the new organisational structure of the hospital which encourages autonomy of the Clinical Directorates. One way of addressing this problem is to integrate both hospital-wide and local systems at the level of the clinical directorate. Although each directorate would have a slightly different 'virtual' computer system, there would presumably be much in common between them. The abstract virtual system that could be shared by all directorates was termed the Directorate Information System. Two projects were set up to address issues in this area - an *ad hoc* reactive one and a small scale experimental, but formal one. It is this latter that is the subject matter of this thesis. Finally some possible functions of a Directorate Information System were presented to give some form to the discussions that are to follow.

In the next chapter, we will investigate the major areas that the experimental Directorate Information System project could most usefully address. We do this by recognising that the cause of the extensive failures in both clinical and business computing is generally considered to be poor systems analysis and in particular requirements elicitation.

[ii] All these activities are explained more fully later in this paper

Chapter 4: Problems with Developing Clinical Computer Systems

4.1 Introduction

Clinical computer systems seem difficult to design, and there have been a number of high profile and expensive disasters relating to their introduction in recent years. Although computing in the NHS appears to be particularly accident-prone, the frequency of computer system failure in all areas of their application should lead us to believe that the design of such artefacts is especially difficult. Studies have shown that this is indeed the case, and the failure rate of computer system developments runs as high as 95%. The most difficult information systems to design are those that will support co-operative work such as is the essence of an organisation - in other words organisational information systems are even more difficult to design and implement successfully than other forms of computer application.

The blame for these failures is accepted by many in the computer industry as due to inadequate systems analysis. In particular requirements analysis has been described as the "hardest part of building a software system" and that "No other part of the work so cripples the resulting system if done wrong.".

Although always difficult and important, aspects of the particular problem being faced can ease the analysis process by providing us with useful guides. In the case of the project reported here the nature of the problem being addressed is such that the computer system designed is to be an 'information system'. One of the assumptions underlying this work (albeit justified at some length below) is that in use, an information system is interpreted into the world as it is perceived by its users. If this is the case, then in order to be acceptable to its users, an information system must be capable of being thus interpreted: it must be a good representation of the perceived world (also called the domain in this thesis). One of the roles of an analyst is therefore to create a description of the domain to determine to what extent an information system can be thought of as such a representation. Having done this, we can make alterations to the design of the system so it is an improvement as far as representational adequacy goes. This is the methodological framework within which the analysis conducted fits. The crux of the problem now is the derivation of a good and insightful description of the domain. This seems particularly hard for clinical information systems as clinicians and computer professionals tend to think very differently about the world - the former anecdotally, the latter universally. The two groups play very different 'language games'. With regard to this, the author turned to the well established scientific method to see if it could shed any light on the problem at hand.

The method of empirical science, as described by Sir Karl Popper can be used to guide the construction of theories of a defined subject of interest. It encourages us to attempt to refute our theories by searching for counter-examples, and to construct ever bolder theories more susceptible to such refutation.

The scientific method has been shown to be enormously powerful and has significantly affected the shape and evolution of society. Although the subject matter of our investigation - the clinical directorate - is 'softer' than the typical target of conventional science, we can nevertheless use the method to great effect. The use of mathematics means that we can investigate the consistency of the theory - something which is vital if we are to build a computer system based on our findings. The bolder the theory around which the computer system is built, the greater the semantic content that system is likely to have, and the 'better' it will be. Through the conduct of experiments, carried out as interviews with users, the theory will become more robust and a better reflection of the reality that those users perceive. Finally, because the end product of the method is a theory constructed of universal statements, and yet the constructive process

involves anecdotal refutations, the method is a particularly good bridge between the community of clinicians and the community of computer scientists.

By the end of this chapter, we will have argued for a particular methodological framework within which the analysis for a Directorate Information System will be placed, and against which a number of existing systems analysis methods will be judged.

4.2 The Difficulty of Information System Design

4.2.1 Clinical Computing: A History of Failure

Computing in the medical sector has been characterised in the last few years by a number of high profile disasters. Although expensive clinical computer systems that provide no great benefit to medical care have been a part of the NHS for many years, the increasingly vast quantities of money being spent on such systems (one survey puts this at £900 million each year [CompAug92]) allied with a willingness by governmental and parliamentary bodies to discover and expose public sector profligacy has illuminated the enormous extent of the problem. In the words of one MP, the NHS is currently plagued by an 'apparently endless string of computer purchase scandals' [CompNov92]. As well as the dubious merit of the HISS programme that has been discussed above, there have been a number of extremely public failures of computer system procurement in the last few years.

Perhaps the most shocking of these was the introduction and subsequent withdrawal after a matter of weeks of the London Ambulance Service's computerised dispatch system [CompApr92], [SWTRHA92]. This system was to be used to log and prioritise emergency calls and control the subsequent response by the service's ambulances. In the short time that the system was operating, many ambulances were incorrectly or inefficiently allocated, and some calls were 'lost' altogether meaning that several emergencies were responded to hours after the initial call. The ensuing chaos almost certainly resulted in several deaths.

The abandoned Healthtrac system, although not responsible for any deaths, nevertheless resulted in large amounts of public money being wasted. West Midlands Regional Health Authority commissioned the development of this system to automate the ordering and purchasing of supplies. It was estimated that £40 million would be saved by the system each year. In fact by the time the system was complete, the process it was to support had so completely changed as to render it utterly useless. This was not before £2.3 million had been spent on the project however [Collins92].

The most expensive scandal concerns the so-called Regional Information Systems Plan (RISP) at Wessex Health Authority. RISP was a strategy which envisaged a coherent set of compatible systems being used at all levels across the region. By insisting on strong central control, compatibility of local systems could be ensured, and the running of the service at Wessex made significantly more efficient. The reality has proved to be very different from this ambitious plan. The authority was investigated by the district auditor who found that it spent over £43 million before abandoning the project, of which £20 million was estimated as having been entirely wasted [CompJul92]. Irregularities in the awarding of the contract only heightened the public's sense of outrage at the waste of taxpayer's money.

These recent failures are only the most visible manifestations of a problem that seems to be endemic in the health service. In 1992 a television documentary (Dispatches) investigated computer related waste in the NHS (this is described in [Collins92]). In the three months it took to prepare the film, the research

team found so many examples of misspent money that it could not include them all in the 45 minute programme.

St Thomas' Hospital is not immune to the condition of misguided computer purchase. For example, the local regional health authority provided a system called the IRC - PAS (Inter-Regional Collaboration - Patient Administration System) to the hospital in the mid-eighties. Although free to the hospital, it cost the health authority millions of pounds. Most of the functionality that this system provided already existed in hospital's other systems. Consequently the large ICL mainframe that the IRC-PAS was implemented on was only used to reformat one file retrieved from the hospital's operational systems so it could be used by the regional health authority's own computer programs.

Even at the departmental level we do not have to look too hard to find examples of money spent on computer equipment that has turned out to have been wasted. When the hospital first introduced clinical directorates in its own resource management project, a commercially available case-mix system (see above for discussion of the purpose and function of this) was purchased to provide the directors with the information that they needed. In the Diabetes and Endocrine Directorate this system was found to be cumbersome to use and not well suited to the idiosyncrasies of diabetic care. The software was delivered to the department on a high-powered (for that time) personal computer which proved very useful as a replacement for the ageing machine used by one of the secretaries. Thus a system that cost tens of thousands of pounds was used as a word-processor.

Some might say that the anecdotes related above are evidence of widespread corruption and incompetence in the NHS. The author's experience is completely contrary to this: the vast majority of workers in the health sector are intelligent, hardworking, and have great integrity. In addition there is a general recognition of the great benefits that computerisation might bring, reinforced by the existence of a few brilliant systems that are used continuously and gratefully by health care professionals. Examples of these are the hospital's 'Telefile' system, and the Diabetes and Endocrine Directorate's Diabeta system. These notable successes do not cover up the vast extent of inefficiency relating to computer system purchase in the NHS, however. If not stupidity and greed, what are the causes of the ubiquitous waste that seems to characterise information technology in the UK health sector? If we are to answer this question, we need to look at other sectors and their experience with computer systems.

4.2.2 Organisational Information Systems: A Tough Nut to Crack

Computer technology is perceived by many as a readily available and straightforward solution to many of the information handling problems facing organisations and societies today. However, it is often the case that the attempt to introduce a computer system creates as many problems as it solves, and large quantities of money can be spent discovering this. If we look at the trade journal where most of the NHS related examples above were reported (Computing), we see that computer fiascos are certainly not confined to the health sector. Dipping at random into issues released over the last year, we find the following headlines.

- Year delay to crime system [McNevin93]

This article reports a delay of a year and significantly increased costs to a £1.5 million integrated police information system. The hold-up is blamed on delayed and complex requirements specifications.

- MPs attack Department of Employment over IT waste [Hill94a]

The Public Accounts Committee (PAC) issued a report [PAC93] criticising a major IT project commissioned by the Department of Employment. The department paid for the development of a system to link training facilities. When the government changed the mechanics of training sponsored by the department, the system was rendered effectively useless. By this stage, the department had spent £48 million on the system.

- NAO finds fresh errors [Hill94b]

The National Audit Office (NAO) issued a report [NAO94] in which £44 million of computer related waste is catalogued. This was not due to the rejection of an expensively developed system, but rather because the system was used and malfunctioned, causing £44 million of benefits to be lost.

The above examples of failed information systems should not lead us to believe that such waste only exists in the public sector. One of the most spectacular computer project fiascos of recent years was the stock market's TAURUS project. After 4 years and more than £75 million spent on development, the system was abruptly abandoned. This was in spite of the recognition that such a system (to settle shares on the day of issuance) was considered vital for the City of London to retain its pre-eminent position in international finance (the City continues to function nevertheless).

The tales of computing débâcles given here, and the fact that they were so easy to find, should lead us to suppose that there is something extremely difficult about the business of designing and implementing information technology systems. In fact, the rarity of good computer systems and the difficulty inherent in building them has been well known to specialists for some considerable time. Fifteen years ago, the US Government Accounting Office completed a study into the worth of nine computer projects which cost a total of $7 million and reported on its findings [GAO79].Of all the software developed, only 2% (assessed on cost) was used as delivered, a further 3% was used after minor modifications, 20% used after extensive modifications, and 75% never used at all. Bickerton [Bick92] describes a more recent study [Pye88] conducted by the Department of Trade and Industry. Here the situation was generally improved in the main, but some types of computer system were much more susceptible to failure than others. In particular, where the process being supported was deemed to be simple (word-processing, for example), the software, some of it functionally extremely complex, was generally successfully implemented. As the work being supported by the system became more complex, so the chances of system failure increased (even though, in this case, the software was not as functionally sophisticated). The study found that none of the software that was developed to support the totality of a particular form of co-operative work (in this case a police investigation system) could be considered a complete success.

What these two sources and the above examples tell us is that developing computer systems is difficult: developing successful organisational information systems that support an organisation's collaborative processes (such as would be the case for a directorate Information System) would seem to be almost impossible. The next section describes what is commonly thought to be the cause of this difficulty - the elicitation and recording of user requirements.

4.2.3 The Essential Problem - Requirements Analysis

Over the last 20 or so years, the nature of computing has changed enormously. Technological advances have been made on all fronts. The main processing units have become faster, smaller, more robust and vastly cheaper. Storage devices, although not much different in type, have hugely improved performance. Whereas computer terminals were definitely the preserve of the computer specialist in the 1970s, now they surround us, and the sight of a high powered computer on everyone's desk is not unusual in a place

of work. Large organisational networks are now commonplace, meaning that office automation, electronic mail, shared word processed files and the like are no longer the preserve of the large computer companies (such as IBM) and their clients, but can be seen in relatively small businesses. In short, the technological advances made by the computer industry are quite staggering - it has been claimed that if the progress made in this field in the last two decades had been applied to car manufacture, a driver could drive to the moon and back in half an hour on a gallon of petrol, in a Rolls-Royce that had cost him or her less than 5 pence and was the size of a matchbox.

Software technology too, continues to improve. Fourth generation languages allied with relational databases support the rapid development of complex applications. The emerging object-oriented programming paradigm will (it is hoped) lead to reusable software components, increasing still further the ability of programmers to develop reliable systems cheaply and easily.

The hardware on which computer systems run is enormously capable, and the software to run on that hardware is easier to build than ever before. As we have seen however, the rejection rate of computer systems by their putative users is still alarmingly high. This is due to the one aspect of applied technology that has hardly improved at all since computers started to be introduced into businesses: the matching of the abilities of the technology with the needs of its users. This process is known in the computing sector as requirements elicitation or requirements engineering. Many commentators consider requirements engineering to be the most critical issue for the discipline of software engineering in the 1990s [Siddiqi94], and yet it is an area where very little progress has been made [Davis94].

Indeed, that the critical problem facing software engineers is systems analysis, which incorporates requirements engineering, has been generally accepted by information systems specialists for many years. One such specialist is Frederick Brooks, who was the manager of the IBM Operating System/360 project in 1964 and 1965. Brooks wrote about his experiences in a seminal work: 'The Mythical Man-Month' [Brooks82], first published in 1975. Brooks considers that software engineering, especially of so-called 'large systems', is one of the most complex technological processes that we can embark on, using the metaphor of the prehistoric 'tar pits' to convey the insidious and inexorable nature of the problem:

"No scene from prehistory is quite as vivid as that of the mortal struggles in the tar pits...Large system programming has over the past decade been such a tar pit, and many great and powerful beasts have thrashed violently in it. Most have emerged with running systems - few have met goals, schedules and budgets ... Everyone has been surprised by the stickiness of the problem, and it is hard to discern the nature of it.".

He claims that to make any progress, the design or architectural issues must be considered in isolation from those pertaining to implementation:

"I will contend that conceptual integrity is *the* most important consideration in system design ... The separation of architectural effort from implementation is a very powerful way of getting conceptual integrity on very large projects."

where

"by the detailed architecture of the system, I mean the complete and detailed specification of the user interface.".

The specification of the user interface is the job of the systems analyst. Twelve years later in 1987, Brooks wrote another highly celebrated paper: 'No Silver Bullet' which developed this theme further [Brooks87]. In this work he is more specific: not only is systems analysis a separate and intractable part of software engineering, but the most difficult and important part of this is requirements engineering:

"The hardest part of building a software system is deciding precisely what to build ... No other part of the work so cripples the resulting system if done wrong.".

Brooks' thoughts have since been adopted and reiterated by many others in the software engineering community, and are now a part of the discipline's orthodoxy. We are thus in good company when we observe that requirements engineering is the crux of the matter, and it is here that we should focus our attention if we are not to repeat classic mistakes.

4.3 Systems Analysis: A Possible Approach

4.3.1 Introduction

In this section we will consider the nature of information systems and see how this should influence our attitude to their analysis. In particular it is argued below that when in use, information systems are interpreted by their users into the organisation being supported (also called the domain). Having accepted this assumption a methodological framework that reflects it can be stated. This requires the representation of the domain in order to judge the adequacy of an information system that is to support it. This is a difficult problem in any domain, especially the clinical one where the barriers to communication are particularly insurmountable. It is explained that the scientific method (briefly described below) might be of some benefit in attacking this issue.

4.3.2 Information Systems: What Are They?

Information systems can be thought of as general purpose tools for supporting the collaborative aspect of an organisation. Thus one of their distinguishing features is that they are used by a number of workers, with a number of tasks. Depending on how we choose to consider the organisation, we can divide the workers into two groups: operational and managerial. In this classification scheme, each group uses information systems in a different way (though we must recognise that one person can act at different times operationally and managerially). The essential difference between these two groups is that operational workers perform the functions that enable the organisation to achieve its purpose (be that producing widgets or curing patients), whereas managers observe this operational behaviour and intervene so as to render the organisation more efficient, profitable, effective, or generally successful. If we use the analogy of a rowing boat, the operational staff would be the oarsmen and women, while the managerial staff would be at the helm, steering the boat. This is of course an extremely simplistic view of the enormous complexity of human organisations (which is not the main subject matter of this thesis): the reader is referred to Morgan's excellent book: "Images of Organisation" [Morgan86] for a thorough and insightful study of the nature of management and organisations.

From this description, we can see that both groups of workers need to be aware of what is happening in some or all of the rest of the organisation and / or its environment to be able to do their jobs. The operational worker is part of a larger whole, and it will almost certainly be the case that he or she will need to be informed of events that have happened elsewhere in the organisation or outside it. Thus the purchaser at a factory needs to know what has been received at 'goods inward' and which finished goods despatched by 'shipping' in order to procure the correct quantities of raw material for the next week. The

telephone receptionist at a hotel needs to know who has called, what sort of rooms they require, and for when, along with the availability of rooms on that day (the rooms possibly having been booked by other people), if he or she is to be able to book the customer's reservation. The doctor in a hospital needs to know what is wrong with the patient and what other doctors and nurses who have already dealt with him or her have discovered and decided if the treatment prescribed is to be successful. It is equally clear that workers in the managerial group need to be aware of what the operational staff are doing, and what the organisation's relation is with the outside world, if they are to be successful at the helm.

An information system helps both these groups of people by presenting them with the information they need to do their work in the form of a description of the current (and past) state of the organisation. In the factory example, a 'Stock Ordering System' might record how much raw material was received today, how many finished goods despatched, and what is on order and when it is scheduled to be delivered. At the hotel, a 'Customer Booking System' might record the rooms booked for other prospective residents, their state of confirmation, and details of the customer in question (such as whether they are new or regular visitors). At the hospital, a 'Clinical Record System' might record the administratively relevant details of the patient being considered and medically relevant ones such as past diseases, conditions, interventions, test results and preliminary diagnoses. In each of these cases, the information in the information system would help the operational worker do his or her job and, in aggregate, would help the managerial worker monitor the progress of the organisation in such a way as to be able to exert control successfully.

All these information systems have in common the feature that they record and represent some aspect of the organisation and the world it is working in. Operational workers are presented with information about that part of the world that is directly pertinent to the task they are engaged on and is available on the information system. Managerial workers are presented with aggregated summary information describing the current and past states of the organisation and its relation with its environment in a simplified, less detailed form.

As a result of the operations the operational workers decide to effect, the state of the organisation is changed (a new order has been sent to a supplier, a room is booked, a patient is admitted to hospital). As the information system will be used again, by the same or another worker, this change should be recorded. We can say then that the operational worker manipulates the state of the information system in accordance with his or her manipulation of the state of the organisation and its relation with the rest of the world. Managerial workers do not manipulate the state of the information system directly - they observe it, and affect the operations that the operational staff engage in as a result of their derived understanding of the state of the organisation and its relation with its environment.

Of course the above description is a vast simplification of the various phenomena that can be observed in an organisation: the division between managers and operational staff is often extremely blurred; the influences on the behaviour of all workers are myriad - many are unrecorded or unrecordable; the idea of an organisation having a state at all is a philosophical presumption that can be easily challenged. Nevertheless, thinking of organisations in these terms gives us insight into the nature of their information systems and enables us to proceed, though we must bear in mind that the philosophical underpinning of our work is based on a simplification. Further ramifications of the assumptions made here are presented in Section 13.2 and Section 13.3.

If we accept the arguments presented above, we are led to conclude that an information system is a representation of some aspect of the world. That part of the world that an information system represents is called its domain in the body of the thesis. In order for an information system to work successfully, its

various users have to recognise the domain in the numbers, words, and symbols presented to them by it. In other words, the users of an information system must be able to interpret the information system into the world. This interpretation must cover the state of the information system (which must describe the state of the domain), the allowable operations on the information system (which must describe relevant operations in the domain), and the state of the information system after those operations (which must represent the state of the organisation after the domain operation that the information system operation is perceived as representing). If the representation of the domain is faulty, then some or all of these interpretations will become difficult and counter-intuitive, leading to dissatisfaction with the system.

Of course there are many other causes of user dissatisfaction, and an information system that faithfully represents the domain to its users might still be extremely unpopular (it might be slow, result in job losses, support an insufficient part of the organisation by having an inadequate domain, or simply annoy people by introducing technology where it is not wanted). These other problems, although extremely important, are not considered directly here: we cannot hope to solve all the problems of computer science in a single piece of work. Indeed we cannot really expect to solve any, but merely address a small range of issues that are of some significance to the development and use of successful computer systems - this is what this thesis attempts to do.

4.3.3 A Methodological Framework

If we accept the argument presented above, we are bound to try and create an information system that can be interpreted into the domain. This thesis works from the assumption that the argument is valid, and the necessity of creating a system that can be interpreted into the domain is one of the main postulates of the work.

In order to create such a system, an iterative engineering stance must be adopted. We first posit an information system design, consider how it might be interpreted into the domain, and make improvements to the design in this light. We cannot investigate this interpretation directly - we generally do not have direct access to the information system if it has not been constructed, and we certainly don't have direct access to user's perception of the domain. We can however, represent the important properties of both of these in a more or less accurate way: we can describe salient properties that the information system and domain are thought to exhibit and investigate whether these descriptions are accurate. Once we have a (to us) satisfactory description of both information system and domain, we can compare them to determine how one might be interpreted into the other.

This argument can be considered as a methodological framework for information systems design, involving the following steps.

- Derive description of the properties of the domain
- Derive description of the properties of the posited information systems
- Inspect the possibility of good interpretation of an information system answering to the derived description into a domain answering to the derived description.
- Alter the design of the information system as a result of this inspection.

Description, or 'reverse engineering' of information systems is relatively straightforward. If we are to understand how this might be interpreted in use, we need a good domain description. Before we consider

how we might get such a description we ought to be aware of some of the difficulties associated with describing and representing other people's perceptions of (an aspect of) the world they live in.

4.3.4 Clinical Systems Analysis: A Breakdown in Communication

The recognition that the majority of the effort should (at least initially) concentrate on domain description rather than any other aspect of the systems development process only helps us so far. This is certainly not the first project to recognise the importance of specifying requirements, and the understanding of the difficulty of the task has not made it any easier or helped to make the success of large computer systems more assured. Specifically we have not seen why clinical information systems seem particularly difficult to deliver.

In the Diabetes and Endocrine Directorate, one of the prevailing opinions at the time of the project's start was that clinicians and computer professionals did not and could not understand each other, the two disciplines speaking fundamentally different languages. This accords with recent developments in philosophy (for example [Lyotard84] and less recently but more seminally [Witt53]) and philosophically informed discussion in the systems analysis literature [Goguen92], [Floyd91].

One of the ideas pervading much of this literature is the rejection of 'classical' notions of objectivity and reality, and their replacement with various forms of relativism. One of the components of this relativism is the concept of the 'local language game' [Witt53]. This contends that the concepts that a person uses to understand their reality are particular to that person, but are shared to a greater or lesser extent with others with whom that person has an empathy, such as friends, relatives, work colleagues, or members of the same discipline. The concepts are shared through the mechanism of the language game within which discourse takes place. Each of the social groups that a person belongs to will have its own language game which that person may be able to play more or less well. Communication between individuals in different groups is only possible inasmuch as the two are part of a larger group (such as both being workers at the same hospital, citizens of the UK, or even human beings) which plays a much looser and less specific language game. The above is a greatly simplified account of this subtle and sophisticated idea. For a richer understanding of this fascinating area of philosophy the reader is referred back to the source documents. [Floyd91] discusses these and other ideas from the point of view of requirements engineering and systems analysis. The idea of local language games is similar to that of experiential reality which is presented in a digestible form by Lakoff [Lakoff87].

We can get a better understanding of the idea of language games by considering two examples - one trivial and one more serious. The trivial example is one that many people will be familiar with - the dinner party. In the author's experience, it sometimes happens that there are two or more identifiable groups of people at the same dinner party in the hope of generating interesting conversation. For example the host might choose a number of his or her guests from the set of university acquaintances and a number from the set of work colleagues, or some from a hobby group and some from friends of the family. If the language games played by guests from each group are specific to that group - for example the university friends reminisce about their shared experiences, and the work colleagues discuss the latest political development in the office - then the discourse between the groups will be minimal and the dinner party will (short of providing sustenance) have failed to achieve its purpose. If one of the groups is a singleton group (that is, it only has one member), then that group may well enjoy no language game at all resulting in an unsatisfactory evening for its member.

A more serious example of the lack of communication between two groups can be taken from the development of computer systems. Here, one group is those who commission and will eventually use a

computer system, and the other those that are responsible for delivering it. The former group is generally referred to as users or stakeholders, and the latter as computer professionals, or more specifically systems analysts. Each group of users will typically play a different language game all of which will be different from the language game played by the systems analyst. One can imagine degrees of 'difference' between the language games played by two groups[iii]. If the first group consists of mathematicians and the second of systems analysts, the difference between their language games is relatively small - the members of both groups place great store by logical thinking and the development of predictable mechanisms, the terms used are often the same (at least at the arithmetic level), and the backgrounds of the members are often similar (many mathematicians, or mathematically inclined people get involved in computing at some level). Possibly as a result of this, much mathematical and scientific software has been extremely successful. If the first group consists of clinicians, and the second of systems analysts, then the difference will be much greater. Not only will the terms used and the backgrounds of the members of the groups typically be very different, but so are some of the most fundamental reasoning structures that each uses.

The end result of systems analysis is generally a system specification. Such a specification can also be regarded as a theory of how the completed system will work. This theory records rules which, assuming that the system is implemented correctly, will always be obeyed. As the rules will always be obeyed, we can call them universal statements, or simply universals. Computer programs themselves are effectively sets of universals - as programmers we cannot (for any reasonably complex system) predict all the behaviours of the system and list a response for each one - rather we describe the required response for a (possibly infinite) range of conditions. Clinicians on the other hand rely on exemplary and anecdotal evidence to great extent. To this extent their job is similar to that of a lawyer. Both professions resort to cases and recalled examples (called precedence in the case of the legal profession) to guide and justify their actions. Groups which play language games incorporating universals will have great difficulty entering into discourse with groups playing predominately anecdotal language games. This incompatibility of language games may go some way towards explaining the communications breakdown between clinicians and systems analysts that has been observed in the Diabetes and Endocrine Directorate. Without effective discourse with the users, the analyst cannot know whether his or her ideas about the requirements of a computer system are genuine or fantastic.

In fact, we can be stronger still in characterising the problems presented by language barriers. Not only do computer scientists play a 'universal' type language game, but such universal statements are necessary if a computer system is to be the result - this is because (as we have seen) a computer system is effectively controlled through the use of universal statements expressed within a computer language. Moreover, these universal statements need to be logically rigorous: paradoxes, inconsistencies and ambiguities in program construction translate as errors in the implemented system. Universal statements are needed then: clinicians (or other users) are totally unqualified to construct them however. Although a typical clinician will be very capable of doing and talking about her job, the process of representing it in the form of a paradox-free, consistent, and unambiguous universal statement is not something that their training has shown them how to do. Conversely it is clear that although a computer scientist might be able to construct a consistent universal statement, she is totally unqualified to talk in any knowledgeable way about the clinical domain.

[iii] It should be noted that it is very difficult to talk about these concepts clearly or with philosophical rigour. In discussing the topic, the author is attempting to utilise the language games of those who espouse the idea. The concept of 'degree of difference' with its implications of extent, position and measurability belong to a different language game than does the notion of 'language game'. However, that a phrase is philosophically suspect does not prevent it from carrying some force of meaning and illuminating the concept that the author is trying to convey.

Even if the argument presented here is correct and is one of the reasons for the difficulty of the systems analyst's task, it is not obvious what can be done about it. The conventional process of analysis - the development of universal theories and their presentation to the users for comment - is clearly inappropriate here. We need a better approach that will help reveal the validity (or lack thereof) of the analyst's (universal) ideas and theories without requiring the clinician or other user to construct universal statements (which would be prone to error). The approach adopted in this project was based on the method of empirical science as described by Sir Karl Popper. Before we consider how and why this approach might address the problems discussed, we must understand how Popper's method works in the general case and why it has had such a major effect on the development of scientific knowledge.

4.3.5 The Scientific Method

Science has been practised for many centuries. Many of the early Greek philosophers could be considered to be scientists in that they sought knowledge about the world they lived in. The words 'science' and 'scientist' were first used with their current meanings in the 16th century [OED87] however, although there was much in these early days that we would not now consider to be science at all, such as alchemy and astrology. Over the years, a feeling for what is valid 'scientific' knowlec ge about the world has emerged, and alchemy and astrology are no longer considered valid occupations for serious scientists. Although the extent and soundness of natural science, and certainly its impact on our lives, has increased steadily since the middle ages, even at the beginning of this century there was much debate as to the nature of science, and how scientific knowledge might be distinguished from other forms. Although the debate continues to this day, it has been shaped by one man more than any other - Sir Karl Popper. Popper elucidated what he termed the method of 'Empirical Science', first introduced in 'Logik der Forschung' [Popper34], published in 1934 which was translated into English as 'The Logic of Scientific Discovery' in 1959 [Popper59]. The publication of these two books acted as an epiphany for many scientists because they presented for the first time an intellectual framework that united the diversity of scientific activity[iv] into one coherent and rigorous (at least significantly more rigorous than previous attempts) philosophical structure.

The scientific method is a means of constructing and testing theories which tell us something about the nature and behaviour of some aspect of the world we live in. We can best understand it by considering two directives which, in order to comply with the method have to be followed. The first is that we must attempt to discredit the theory with utmost vigour, and only if we fail in this task can the theory be considered valid. The second is that we must attempt to render the theory as easy to discredit as possible - the 'best' theories are considered those which appear easiest to disprove and yet, in spite of the best efforts of scientists, have not been.

A theory is a set of rules which we claim hold true about some specified subject matter in which we are interested. These rules, in common with most rules, are of a 'thou shalt not' rather than 'thou shalt' form. In this way they describe and prohibit classes of phenomena. The testing of the theory involves the search for (repeatable) examples of the forbidden phenomena. If any are found, the theory is considered to have been refuted, and must be reconstructed.

Although not always the case, for the purposes of the argument we can assume that a scientific theory is presented in mathematical terms: a number of statements are given which outlaw certain classes of

[iv] It must be said that the strictness of Popper's arguments are more easily applied to the 'harder' sciences such as physics than the 'softer' life sciences such as biology (of which Popper himself was unfairly quite dismissive). Even so, students of the life sciences were equally affected - Sir Peter Medawar, the famous biologist, said in a review of 'The Logic of Scientific Discovery' published in the New Scientist: "one of the most important documents of the twentieth century".

behaviour. In mathematical parlance, these statements are called axioms. The advantage of mathematics is twofold. Firstly the meaning of mathematical terms and symbols is very well defined. The 'fuzziness' and 'ambiguity' of natural language is avoided resulting in a clear and emphatic meaning for any statement. Secondly mathematics comes with a deductive mechanism - from the axiomatic rules we can deduce derived properties or theorems which we claim must hold true of the subject (and act to specify further proscribed behaviours). These theorems can in turn be used as the basis for further derivation. It was originally thought that it should be possible to tell from a sound theory whether any given behaviour is allowed: Gödel's famous 'incompleteness theorem' tells us that this is not in general true, but the power of mathematics to create complex intellectual edifices from seemingly extremely simple axioms is great - Bertrand Russell derived (albeit not uncontroversially) almost all of mathematics from three axioms [White87]. The set of behaviours that we can show are valid according to the theory is sometimes known as the theory's consequence closure.

Once we know what behaviours are forbidden by the theory, we must search for examples so as to discredit it. We should not waste our time looking for behaviours that we are unlikely to see - on the contrary we must take care to look for behaviours that we might expect to see and yet are forbidden by the theory. The choice of which behaviours to look for, and how to look for them is the art of the experimental scientist. A scientific theory is always a simplification - we cannot describe the whole world in all its complexity, and the theory thus only lends insight into a well defined part of all natural phenomena. The construction of experiments must thus endeavour to exclude those parts of the natural world that the theory is not concerned with. An experiment which does this is called a 'controlled experiment'. The advantage of the precision and clarity of universal statements expressed in mathematical notation here is that it is relatively straightforward to determine whether or not an observed behaviour does indeed represent a refutation.

The second directive was the need to render the theory as falsifiable as possible. Clearly, the more behaviours a theory prohibits, the more falsifiable it is as there are more refutative counter examples to be found. The more behaviours that are forbidden, the more powerful and useful the theory is. In the body of the thesis, a theory which is more falsifiable than another is said to be bolder than that other: the second directive tells us that if a theory has not been refuted, we should endeavour to embolden it.

The above is a greatly simplified description of the method of empirical science - there are many subtleties not described here but to be found in Popper's original works concerning such areas as probability and quantum physics. Nor can we maintain that Popper's ideas cover fully the concerns of scientists. For example scientists must also be concerned with the expressive elegance of their theories, and the utility of their subject matter.

4.3.6 Why The Scientific Method Will Help

From the above description it should be clear how the scientific method might help overcome some of the problems associated with requirements analysis. In a previous part of this chapter we saw how, if a domain description is to useful in the construction of an information system, it must take the form of a mathematically rigorous universal statement.

This presents two related problems, both of which are addressed by the scientific method. Firstly a domain theory constructed by the analyst will inevitably be flawed - the analyst's understanding of the domain will be very different from that of the user, and will contain many preconceptions and plain errors. Additionally, in the case of the clinician especially, and perhaps also with other users, the 'universal' language of the analyst (needed to derive a computer system design which is a universal

statement) is very difficult to converse in. Clinicians (and perhaps other users) would appear to be much happier talking in anecdotal singular form (this is a simplification - another parallel argument has it that no matter which form of language clinicians use, we can rely on the singular more than the universal).

It is thus not only important, but also highly problematic to root out errors caused by a misunderstanding of the domain on the part of the analyst. The scientific method as described is a likely candidate for a tool to overcome these problems. Firstly, as we have seen, the use of mathematical notation to express theorems of the theory means that it is a comparatively straightforward task to determine whether or not an observed (or as we shall see, related) behaviour is a refutation of the theory. This adds a degree of objectivity[v] to the process of unearthing and challenging the preconceptions of the analyst which have been expressed as a theory of the domain of interest. Secondly the scientific method acts as a bridge between singular and universal statements. It was stated earlier that the user or other stakeholder in a system is generally not capable of expressing a formally consistent description of the domain whereas the computer scientist, while (more) qualified to construct consistent universal statements has little or no understanding of the domain. The scientific method, through the exploitation of the singular statement in the construction of the universal helps us resolve this paradox. The key is that while the universal statement as expressed by a user is prone to errors, the singular statement is much less so. We would be rightly sceptical of a user's claim that 'all clinical appointments apply to a single patient': we would have less reason to disbelieve the statement 'my last appointment was for Mr Jones and no-one else'. The use of the scientific method does not just facilitate the communication between computer scientists, but more importantly it enables the refinement of universal statements so that they do not conflict with the experiences of the domain workers.

The situation is not quite as simple as has been suggested however. The subject area, or domain, that we are interested in is not generally investigated using the method of empirical science, the approach is more suited to 'hard' physical systems and is weaker when it is used to investigate the nature of soft of human organisations. However, as an additional tool it can be used even in this unlikely setting and will shed valuable light on the nature of the problem domain. The way in which the method was harnessed to create a theory of a clinical directorate is presented in the next section.

4.3.7 How Might The Scientific Method Be Used Here?

There are a number of ways in which the scientific method can be exploited to support the analysis needed to construct an information system. Firstly scientific theories must be internally consistent if they are to have any validity at all. The consequence closure of an inconsistent theory contains negations for all its theorems - in other words anything the theory prevents it also allows and *vice-versa*. If we can derive any theorem of the form 'statement A is true and statement A is untrue' then the theory is inconsistent and must be re-constructed. Similarly, a valid computer programme specification must not be inconsistent - if it were it would not be capable of being implemented.

The directive which tells us to embolden the theory is also useful in the development of computer systems. The data held in a computer's database can be described in terms of invariants which act to forbid certain classes of behaviours (it is in this sense that a specification is a theory). The stronger the invariants, the bolder the specification, and the more information content the system is considered to have [Cohen92].

[v] Although as we shall see in Section 13.3 the process is still highly subjective: it is nevertheless less subjective than many of the other approaches commonly used in requirements and systems analysis.

Most importantly perhaps, we can refute the theories that we have used to describe the domain. We can look on discussions with users as experiments (although these are admittedly not rigidly controlled in the way a controlled experiment in the natural sciences would have to be). Interviews can be held with the users and other stakeholders covering topics that the analyst wants to explore - areas where the theory presents surprising results or where it is anticipated that the theory must be particularly accurate. The analyst searches for counter examples to the theory in the responses that the interviewee makes: if such a counter example is found, then (assuming repetition, perhaps in the form of agreement from a second interview with a different interviewee) the theory has been refuted. In this way the theory acts as a form of script around which the analyst must base conversations and interviews with workers in the domain. It is moreover claimed that this method of creating universal theories is particularly well suited to the problem of linking the players of a language game which is strongly anecdotal (such as clinicians, or so it is claimed) with players of one that leans towards the universal (such as computer professionals). The facilitation of communications between analyst and domain worker is beneficial, and can lead to an accurate domain description expressed in the form of a consistent universal statement.

Before we blindly accept the assertions of the above arguments we should be aware of a number of problems. These are discussed later in the thesis in Chapter 13: here we will consider some of them briefly so that we can view the results with the appropriate level of caution. Popper's 'Logic of Scientific Discovery' was the source quoted at the introduction of the concept of the universal and singular statements. However, in the same work Popper also notes that the two are not as clear cut as we might suppose, and that embedded in every singular statement are fragments of the universal. The use of any term implies the existence of some underlying universal concepts - use of the word 'doctor' for example implies the existence of a group with this name, distinguishable from others. Science, it is claimed does not then rely on the rigid distinction between universal and singular statements as is illustrated in the following passage taken from the book:

'*The empirical basis of objective science has thus nothing 'absolute' about it. Science does not rest on solid bedrock. The bold structure of its theories rises, as it were, above a swamp. It is like a building erected on piles. The piles are driven down from above into the swamp, but not down to any natural or 'given' base; and if we stop driving the piles deeper, it is not because we have reached firm ground. We simply stop when we are satisfied that the piles are firm enough to carry the structure, at least for the time being.*' (taken from [Popper80] pp 111)

Once we have seen that the distinction between universal and singular statements is somewhat artificial we can consider the language games played by clinicians and analysts in a sophisticated way. While both clinicians and computer scientists use universal statements, those of the latter group are more universal than those of the former group. That the statements of the clinicians are less universal than those needed to make a computer system means that the problems of inconsistency ambiguity and paradox are lessened, not removed. If there is a refutation of the description or theory that has been constructed we should not assume that it must be the theory that is at 'fault': the problem might lie in the user's expression of her experiences.

There are a number of ways that we might choose to deal with such an 'error' in the user's understanding. Firstly we might consider that a given 'disagreement' with the theory has been caused by a genuine confusion in the mind of the domain worker over the nature of the tasks in which she engages. In this case, the theory would be considered to be 'correct' and the possibility thus presents itself of 'educating' the user, improving her understanding of her job. The process of client education is central to the Total Quality Management (TQM) and Business Process Re-engineering (BPR) approaches to planned

organisational development. Although not the subject matter of this thesis, these disciplines are important to the evolution of information systems as a whole, and are briefly discussed further in Section 14.7.

Another possibility is that two sub-domains are perceived according to different concepts and 'internal models' by a domain worker. This is often the case, even in the 'hard' sciences, and is a useful intellectual tool in the attempt to render an extremely complex world comprehensible. Although mutually inconsistent perspectives of reality might be helpful to us humans in understanding the world, they are fatal to computer systems. In this case, user education is not appropriate as the understanding of the domain is already adequate. The best we can hope to do is discover where inconsistencies exist, and where they do, to discover which of the two possible theories is more likely to represent the way the user will view the world in these circumstances.

In general however, we as analysts should have the humility to recognise that the user knows her domain better than we, and that if a disagreement between the domain worker's account and the theory is observed, the error most probably lies with the theory and not the user's understanding. We should not discount the latter possibility, just recognise that it is less likely than the first.

4.4 Conclusion

In this chapter, we saw that the design of computer systems of all types - especially those in the clinical sector - is extremely difficult and prone to failure. Especially difficult are organisational information systems such as the proposed Directorate Information System. We saw that the culprit for this difficulty is the initial systems analysis, and especially that part responsible for specifying the requirements of the system, known as requirements engineering. It is argued that all information systems are interpreted as aspects of the domain that they are supporting. This leads to the statement of a methodological framework for designing systems that can be interpreted into the domain. Use of a method conforming to this framework creates the necessity of a good description of the domain. It is the process of getting this domain description that is at the nub of requirements analysis, and it is here that the essential difficulty lies. The apparent intractability of understanding the user's perception of the domain is to a great extent caused by the different and incompatible language games played by the potential users of the system and its designers and implementors. This difficulty is especially pronounced when the users are clinicians. The use of the scientific method in the elicitation and representation of requirements was described, and it was explained how it might help address the problems associated with requirements engineering in general and requirements engineering in the clinical sector in particular.

We are now in a position to refine the methodological framework so that it reflects the one that guided the project reported here.

- Derive a description of the properties of the domain, using the scientific method so to do.
- Derive descriptions of the properties of the posited information systems
- Inspect the possibility of good interpretation of an information system (conforming to the derived description) into a domain (conforming to the derived description).
- Alter the design of the information system as a result of this inspection.

The first stage, the derivation of properties of the domain, depends crucially on the use of the scientific method with its insistence on theory construction and refutation, and formalism of representation.

The next chapter will briefly consider some of the conventional systems analysis methodologies in the light of the discussions above, and describe in more detail how the scientific method might be used to address the problem at hand, and in particular what theories we should construct.

Part Two:

Review of Methods, Method Used and Identification of Problem Boundary

Chapter 5: Review of Existing Analysis Methods

5.1 Introduction

In this chapter we will explore various existing methods of systems analysis in the hope that one will be capable of exploitation within the framework we have judged to be appropriate for the project. We first consider analysis in general: what are the features that are common to all techniques of analysis and set them apart from other forms of problem solving? It is argued below, and illustrated with two examples taken from disciplines other than computer science, that these common features are threefold: the set of problems capable of resolution through the use of a particular process of analysis tends to be small and well defined; a limited set of procedures is available to be used in solving the problems; and the results of the analysis are presented in a standard form. The type of analysis we are interested in, systems analysis, is no different. Each method of analysis addresses a well defined type of problem, each presents its would-be users with a set of techniques and procedures to use, and each encourages those users to record the results of the analysis in standard forms.

From the observation that all analytical techniques consist of processes and presentational media, and that the systems we are analysing should represent a part of the world, we can derive three criteria to judge the methods being reviewed. These are:

- do the processes conform to the scientific method?
- are the presentational notations semantically rigorous?
- are the problem and solution distinguished?

This chapter proceeds to criticise a number of analytical methods for failing to meet the three criteria.

5.2 What is Analysis: Process and Presentation

Many disciplines, be they of a scientific (both hard and soft) or engineering nature, use forms of analysis to understand further a well structured problem, and perhaps to suggest a solution. The term has come to describe so many activities that it has become abstract and vague with very broad usage. Indeed, the dictionary entry of the word confirms this - if we look in the Oxford English Dictionary we see that analysis is described as:

'...the exact determination of the elements or components of anything complex':

a definition so imprecise as to verge on the meaningless. Consequently in use the word is commonly used with a qualifier, and even where it is not it is because the type of analysis being carried out is well understood. There is however a common essence to all these forms of endeavour that mark them out from the more general search for knowledge. We can illustrate this commonality through the use of two forms of analysis with which the author has a (very) small degree of understanding: the chemical analysis of metal salts (which many will remember from their school days), and the structural analysis of bridges. The aspects of analysis that are common to all its forms might be considered to be as follows.

- The problem area is (relatively) well defined.

Although analysis as a concept is very broad, each of its multitudinous forms is fairly well defined, and addresses only a small class of problems with well defined boundaries. If we take our example of chemical

analysis, we have selected that sub-specialty which addresses only the composition of metal salts. The analysis method is incapable of identifying and discerning other compounds, so in general it would be preceded by a broader compositional analysis or some other form of reasoning that meant that the analyst was sure that the substance being investigated was in fact the salt of a metal. Not only are the types of substance that the analysis can be used to investigate restricted but the results of the analysis are limited as well. Chemical analysis can tell us the chemical composition of a compound - it will not tell us anything directly about its crystalline structure (though it might be possible to deduce this from its chemistry), its mechanical properties, its cost, abundance, or utility. All these questions can be answered, but each requires a different form of analysis.

Our other example was structural analysis, specifically that relating to static load bearing structures (such as bridges). The form of analysis used by bridge engineers would be of little use for exploring the structure of an airframe as it passes through the sound barrier, or a golf ball as it is hit by a club. The complexity of these problems is greater, and so the analytical methods used are more involved. As with the chemical example, not only is the range of entities susceptible to investigation limited, but so is the range of properties pertaining to those entities. Structural analysis is a useful tool for predicting the stresses that a given structure will have to endure, aspects of the finished article such as aesthetic appeal or traffic carrying capacity are totally outside the scope of the technique.

- There is a recognised and recorded structure to the analysis process

For any type of analysis, there is a limited and well defined set of procedures that it is appropriate to use. In many cases, these procedures must be undertaken in a specific order - in short, there is a structured method to the analysis.

In the case of the chemical analysis, the techniques and procedures at our disposal include: the measurement of solubility in various solutions (usually aqueous); flame tests; measuring electrolytic potential; testing acidity; and in more complex cases various forms of spectrum analysis (infra-red, ultra-violet or nuclear magnetic resonance). As chemical analysis is so established and refined, there is a recognised sequence to the application of each of these processes leading to the most rapid identification of the substance in question. There are other techniques that can be used to investigate the nature of a compound - magnetic tests can tell us whether it is diamagnetic or paramagnetic, x-ray crystallography will help us discern its internal atomic arrangement, and investigation of appropriate catalogues or the commodity markets will tell us of its likely price. All these might be useful but have no place in the process of chemical analysis.

In the case of structural analysis of bridges there are a similarly well bounded set of procedures that the engineer deploys to address the problem. These are essentially the identification of forces applied to the bridge, the resolving of those forces through the structure and the equating of the forces and moments such that there are no net linear or angular forces (which would result in a decidedly non-static bridge). If the deformation of the bridge is important then the restoring force of its component parts becomes significant, and the stiffness coefficients of the materials used in its construction must be added to the equations balancing forces and moments. In general these calculations are sufficiently well understood that they can be performed by computer, using the method of 'finite element analysis' which enables the structure to be considered as a composition of many more parts (meaning that the resolution of the analysis will be finer) than would be reasonable with humans performing the necessary calculations - the resolving and balancing of forces and moments are still the major processes that constitute this form of structural analysis.

- There is a recognised and standard means of representing the results of the analysis.

There is generally a standard notation used for presenting the findings of the analysis, enabling the maximum relevant information to be presented in the most succinct way. In the case of the chemical analysis, the notation used depends on the complexity of the substance that has been analysed, but the compound is simple, the familiar system where the elemental components are represented by letters or pairs of letters, and the molecular ratio of these elements by subscripted numbers is used (thus NaCl for Sodium Chloride, and $KMnO_4$ for Potassium Permanganate). More advanced nomenclature presents the informed reader with more information - thus pseudo-cumene represented in standard S.I. nomenclature becomes 1,2,4 trimethyl benzene indicating that a molecule of this chemical takes the form of a benzene ring attached at its first, second and fourth vertices to a methyl group ($-CH_3$).

In a similar way, structural analysis relies on a limited number of representational notations. The most important of these is the force diagram where the forces, stresses, and strains affecting the structure are represented in geometrical form. In addition to this diagrammatic notation, numeric conventions are used to represent elastic deformations and safety limits (such as the maximum axle weight permitted on such a bridge).

In our case we are interested in that branch of analysis known as 'Systems Analysis', specifically that part which relates to the construction of computer systems. It should come as no surprise to hear that this form of analysis exhibits the general properties described above. As the discipline is fairly young, there are a number of methods vying with each other for primacy. That none is entirely satisfactory can be seen from the large number of computer systems which fail to satisfy the requirements of the user. Each method uses a different set of procedures, and presents its results in a different form. All have the same limited domain of application: this is the construction of the design of an information system starting with a poorly articulated expression of need. In the next few sections we will discuss the process and presentational notations of a number of systems analysis techniques, and consider how well they conform to the methodological framework described in Chapter 4.

5.3 Some Popular Methods

5.3.1 Criteria For Judgement

In the last chapter of the previous part we discussed and stated the methodological framework that will be used in the analysis and design of the information system we are interested in. We can use this to guide investigations of common systems analysis techniques. The first stage of an analysis within this framework must provide a description of the domain of interest derived using the scientific method. The processes associated with the systems analysis technique we choose should thus conform to the scientific method. The success of science has depended partly on the rigour of the notation used to construct the theories being developed. The presentational notation supplied with the systems analysis technique should thus have formal semantics and a deductive calculus. Even if the processes of the analysis technique do not conventionally support the scientific approach, it might be possible to use the presentational notations that accompany it providing they are mathematically sound in this way. Finally, our methodological framework insists that we need to assess how well the IS can be interpreted into the domain. This suggests another property that the analysis technique should exhibit: how well does each ensure that the resulting information system can be interpreted into the domain? If the method does not ensure this, we need to understand the implications. This is in many ways similar to the engineering maxim that the problem must be understood before a solution can be created: in the case of the design of an information system, understanding the problem involves understanding the organisation that it is to serve.

The above discussion can be summarised by stating three criteria that can be used to judge the suitability of systems analysis techniques:

- Do the technique's processes conform to the scientific method?
- Do (any of) the technique's presentational notations possess formal semantics?
- Does the technique encourage the description of the domain separately from the information system - does it distinguish between the problem and the solution?

In the next few sections, we will investigate a few different classes of method. We will judge each of these according to the criteria given. We shall see that, as a result of judgements on the basis of these criteria, all the methods considered are rejected: this is not because the methods themselves are inadequate, but rather because they do not conform with the scientific method, their notations do not support the sort of reasoning we are interested in, or they fail to adequately separate the problem and solution.

5.3.2 Data Modelling

Data modelling is the most common form of analysis used in the design of information systems: it uses such familiar techniques as data flow analysis [Yourdon89], functional decomposition [Yourdon78], entity relationship modelling, [Chen76] as well as subsequent developments [Hull87]. There are a number of distinct methods in use, the most familiar being James Martin's Information Engineering [Martin89], the French MERISE [Flynn93], and the UK government's SSADM [Downs92]. SSADM is typical of all these methods and can be taken as a representative example of the class.

SSADM commences with a feasibility study during which the problem to be addressed is defined and the feasibility of a computer system solution assessed. This stage is followed by a requirements analysis phase which examines the existing information system (automated or otherwise), and considers different options for change to this. The existing information system is described in terms of dataflow diagrams, or DFDs, and logical data models, or LDM (also known as entity relationship - ER - models, entity relationship attribute - ERA - models and Bachman diagrams). The requirements analysis phase is followed by requirements specification. This is a specification of the proposed 'solution' system specified with LDMs and DFDs. The subsequent logical system specification refines this high level design through the use of user dialogue definitions and entity life history - ELH - diagrams. Finally the physical design phase takes decisions pertaining to performance and cost criteria - for example whether a given entity should be represented as a data file in a relational database, or 'hardwired' into the system, and whether it should always be kept in memory, or have to be reloaded from disk each time it is accessed.

There are two things that should immediately strike us reading this (admittedly extremely brief) summary of this very popular method. Firstly almost all of the process is focused on the design of the new system (based on the old system) rather than attempting to understand the problems facing the organisation, and secondly there is little mention of the users of the system being analysed. These two observations are related.

In the standard textbook of the latest version of SSADM, there are five phases as described above. The first is the feasibility study, which is described as necessary, but maybe not as part of the structured part of the analysis. A 'flow chart' is given describing this phase - a simplified version is presented here.

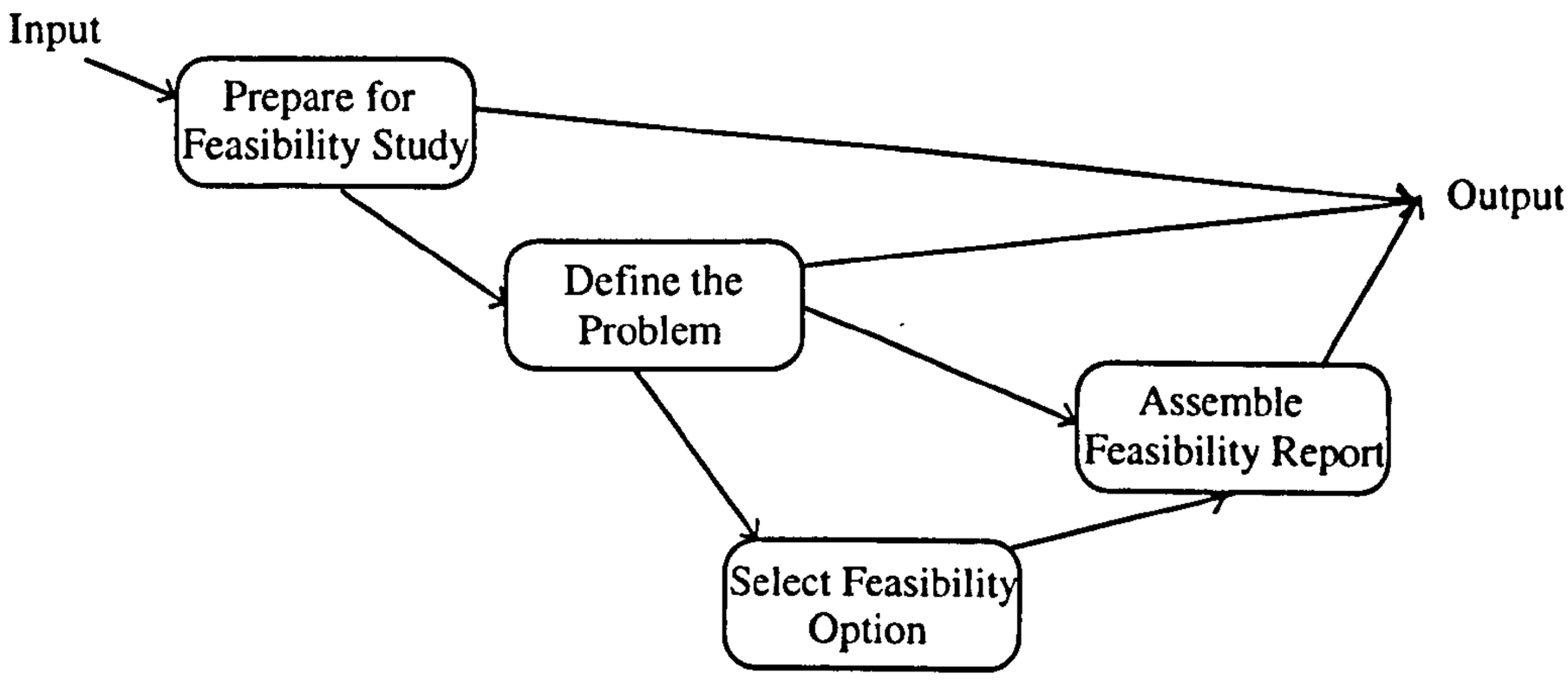


Figure 2-1: Detail of Feasibility Phase (One of Five) of SSADM Method.

This is the only phase in the method where the problem is defined, and here it is only one step of four. It would seem from this that SSADM is best suited to projects where the problem is very well understood already. Some might object to this statement by observing that we have already accepted that the essence of the problem lies in the representation of the organisation, and that SSADM and the other data modelling methods are good at doing this through the use of LDM and ELH diagrams. While it is one of the assumptions of this thesis that representation of the domain is essential to good information system design, it is not clear that SSADM does in fact support this. LDM diagrams, for example, were first described by Chen [Chen76] as a means of developing semantically relevant database systems, specifically those supporting the relational model as defined by Codd [Codd70]. In SSADM, an LDM diagram is derived in the requirements analysis phase and refined in the requirements specification phase. The operations on the entities in this structure are defined in the logical systems specification phase, and their implementation planned in the physical design phase. Thus the LDM diagram is always a (possibly very high level) specification for the structure of the eventual database. The distinction between the representation of the domain and the representation of the information system (which is what a specification is) is not made. This might be a pedantic point if the information system was just an implementation of a specification that was identical in form to an LDM based description of the domain, but this is very rarely the case. As the same description is refined to produce the specification of the database structure, decisions taken early on in the LDM modelling process affect the design of the eventual system enormously. As a result, issues relating to the representation of the domain and issues relating to implementation considerations become unavoidably confused: the method does not in fact successfully distinguish domain and computer system, and so does not manage to separate problem and solution.

The other observation that was made above concerning the method was the lack of communication with users. The majority of the method is devoted to the documentation of the specification of the required system, with very little discussion of how that specification is to be elicited from the stakeholders. Where mention of the difficulty of effectively communicating with users is made, it is as an aside rather than central to the argument. For example, at the end of the description of logical data modelling in one of the standard SSADM handbooks, we find a short 'cautionary note':

"... there are many situations where judgement is needed as to whether or not something is an entity. The general principles are ... firstly consult with the people whose system it is and try to reflect their view of what matters .. secondly to try to make the model describe the underlying and stable rather than the immediate and specific." [Downs92, Page 128]

In the book, this discussion of user involvement lasts for eleven lines! Even if we choose to spend more time talking to users than is suggested by the method, we do that by discussing the models that have been so far devised, be they in the form of DFDs, LDMs, or ELHs. These models are of a universal nature, and it is these universal statements that we discuss with the users, seeking approval for the 'theory' contained within them. This is very different from the scientific approach described earlier, and does not help overcome the language barrier discussed.

To sum up the above arguments, the process of SSADM (and by implication other data modelling techniques) is not scientific (which is one requirement for the method to be used), and it does not sufficiently distinguish between problem and solution (which was another requirement). What of the presentational notations? In common with other forms of analysis, data modelling techniques use certain pre-defined standards for presenting their results. These are many and varied, and in general each method is supplied with its own diagrammatic conventions. Again, SSADM is typical of the field, and inspecting this method will tell us if the notation used for recording the results of the analysis can be exploited by a method of our own construction. Alas, the answer is no: the semantic power of the different representational systems is too poor. Of the three main diagram forms, DFDs, ELHs and LDMs, it is the latter which has the firmest mathematical foundations but even these are poor. Firstly what can be expressed using an LDM diagram is formally extremely limited, and secondly the deductive power we can use on the theorems expressed in an LDM diagram is almost non-existent.

We can understand the first point by considering two mathematical relations: less than (<), and less than or equals (≤). Although these are relations over the same entity - numbers - they say different things. The way we would express the first of these relations is to use a 'pig's ear' relation over the 'number' entity thus:

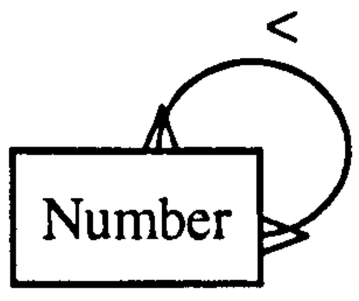


Figure 2-2: LDM (or ER) diagram for the relation '<'

but this is also the diagram we would have to use for less than or equal. This is because all the diagram has said is that the relation '<' holds between any number and some other numbers. In the set theoretic notation used elsewhere in the thesis, we have said:

$< : \text{Number} \leftrightarrow \text{Number}$.

The second point is related to the first - because we have not said very much about the relation, we cannot deduce many of its properties. For example, we know that one of the properties of < is as follows

$a < b \;\&\; b < c \Rightarrow a < c$.

We cannot deduce this from the diagram, because exactly the same pictorial representation (with different labels) would be used for relations that do not have this property, such as 'is a friend of' between people, or 'is adjacent to' between objects. If we could deduce this property, known as 'transitivity', for <, then we could also do it for these other relations. The other notations used by SSADM share similar problems associated with low semantic content.

As the various data modelling methods, as represented by SSADM, do not follow the scientific method, do not sufficiently distinguish the problem and solution, and come with notations that have limited formal content, we shall not use any in the work we are engaged on. Some readers might raise an objection here: 'surely a thousand systems analysts can't be wrong?'. Indeed, data modelling is extremely popular in the systems analysis and software engineering community - why is this? The problems discussed above mean that these methods are weak as requirements elicitation and specification tools: as systems engineering tools they are very powerful however. If the nature of the proposed system is clearly understood, then SSADM and its sister methods are a useful means of engineering an information system from the (already known) requirement. Thus if we look on data modelling methods as guides for solution engineering, and use them as such then their problems are no longer of such concern and their advantages (including lucidity, clarity of representation, ready availability of automated support tools) explain their popularity.

If we are going to use a data modelling method as some form of solution engineering guide, we should be aware that there are differences between their different forms. In particular the recently developed 'object-oriented' modelling systems are supposedly easier to understand (by virtue of class inheritance) and provide better support for the dynamic properties of the system being modelled. Particularly strong on this last point is the method described by Rumbaugh [Rumbaugh91].

5.3.3 Soft Systems Analysis Methods

There are a number of analytical methods that can be thought of as 'soft' in their outlook, but most have been derived from the most celebrated of all of these, Checkland's Soft Systems Methodology [originally Check72 , and more recently Check90]. This approach to problem solving will be taken as a prototypical example below. The benefits of Checkland's method apply to most soft systems methods, and similarly the criticisms of the example can be levelled at all members of the class.

The Soft Systems Methodology was devised by Peter Checkland at Lancaster University. It is very much a product of the 'systems movement' which could be considered to have started in the 1930s by Betalanffy with his 'General System Theory' [Bert68]. This movement considers human systems as holistic entities with emergent properties that can only be understood by considering the whole. The systems approach to problem solving has split into 'hard' and 'soft' camps. The hard methods consider systems that have a clear purpose and well defined goals, and are useful for designing solutions that achieve those goals. The soft methods on the other hand recognise that many human activity systems are so complex that they do not have a single goal, and to impose a solution that embodies a single purpose can be extremely damaging to the system. Checkland's method, which he calls Soft Systems Methodology (SSM), lies in this second camp.

SSM can be understood as having 7 steps. The first is to investigate the problem situation, and the second to express this in the form of 'Rich Pictures' which describe the problem from a social as well as 'mechanical' perspective (an example of a Rich Picture can be found in Section 7.2). Of crucial importance at this stage is the recognition that different stakeholders in the problem might have very different understandings about its nature: some might not consider the problem to be a problem at all. Checkland refers to these as different 'world views' or 'Weltanschauungen'. The investigation of different

world views reveals potential conflicts within the organisation being studied, and these must be recorded on the rich picture. The third step is to devise a number of 'root definitions' that endeavour to describe the essential nature and purpose of the system. It will generally be appropriate to have one root definition for each world view. In the fourth step, for each root definition a conceptual model is created describing a hypothetical system that would satisfy that purpose, and that purpose alone. The fifth step is to compare the various new hypothetical systems with the current perceived reality as represented in the rich pictures produced in step two. This comparison is done in conjunction with the stakeholders of the problem - the main purpose is to stimulate discussion about possible changes to the current organisation. The sixth step is to recommend changes on the basis of these discussions, and the seventh is to implement those changes.

One observation that we can immediately make about SSM is that it explicitly and clearly distinguishes between the problem and a possible solution. Although that solution does not necessarily take the form of a computer system, that is certainly one of the possible results of the decision making process in steps 5 and 6 of the method. For this reason SSM is popular as a method to be used to define the nature of the problem and suggest the scope of a computer based solution. At least one popular method [Avison90] combines SSM and SSADM together such that the first describes the boundaries and requirements of the computer system, and the latter helps in its engineering. However, while the organisation and its associated 'problem situations' are explored explicitly, that exploration is not carried out in a 'scientific' manner such as we require. Indeed, it is the rich picture that is the main form of communication and record during the step which describes the problem domain, and the stakeholders are actively encouraged to take part in its construction. While this is highly commendable, it does not follow the scientific method, and the end result is a diagram that has been agreed rather than tested.

As with the data modelling, we cannot even reject the method but use the associated notations. The only representational notations that are at all standardised are the rich picture and the bubble diagrams that are used to present the conceptual models of proposed systems. Neither of these has any formal semantic content at all to speak of. In fact, the SSM can be followed perfectly well without using either of these notations, relying on conventional English instead[vi]. There is consequently nothing here that we can use to support formal reasoning.

Although we will not use the method because it fails some of the criteria we are insisting the method chosen needs to satisfy, SSM nevertheless raises issues that the analyst must be aware of, and need to be addressed by any analysis. The first is that there is no such thing as 'the purpose' of a complex human organisation: at least not one that we can unequivocally define. The analysis reported in this thesis tried as far as possible to avoid describing any form of purpose at all, concentrating on the operational rather than managerial aspects of the directorate (it was considered that there would be less controversy surrounding what the directorate 'does' than why and how it does it).

The second issue raised that we must be aware of is the lack of any single world view, and the possibility that different stakeholders or would be users have very different outlooks, and 'constructed realities'. This issue is discussed further in Section 13.3, but we should note here that however many different versions of reality exist amongst the workers in an organisation, a computer system can only work according to one consistent set of rules. The theory that we construct can only support a single view of reality: the analyst

[vi] It might be said here that this disagrees with our earlier insistence that all analytical methods presented their results in standard forms. This is indeed the case, and SSM is very unusual in this respect. Some would say however that SSM is not an analytical method at all, but rather a framework for 'Action Research'. If this is so, then we should not be surprised to find that the process is considered of much greater importance than any presentational products, and the result of that process is an altered organisation. This differs from SSADM which places much store on the creation of 'products' such as LDMs, DFDs, and ELHs.

must ensure as far as possible that that theory can support, and is not refuted by, the multitude of experiences reported by the workers interviewed.

A final point concerning SSM is that its use is not confined to designing information systems, and might indeed be better used for other purposes. The way in which different world views are accommodated and different conceptual models compared with the real situation before changes are chosen and implemented makes it particularly useful for organisational restructuring projects. Business process re-engineering, although an extremely interesting subject, was not the subject of the work reported here: the relevant decisions had already been taken by the time the Directorate Information System project started. If the project had started in an even more ill defined way than it did, then SSM might well have been an appropriate tool to use, and the creation of a Directorate Information System might have been one of the recommended changes to be implemented in step seven of the method.

5.3.4 Rapid Prototyping

In many ways, rapid prototyping is not a method at all, but we will consider it here as it is a recognised and popular means of proceeding from a loosely structured requirement to a finished system. There are many forms of prototyping used by the information systems community, ranging from one off prototypes to see if the conceptual system design meets the stated requirements, to evolutionary prototypes that start as mock-ups of the proposed system and end up being used in earnest (see [Boehm88] and [McCrack82]), to 'throw-away' prototypes that are used to elicit requirements from the users and then are disposed of [Dearnley83]. It is this last form of prototyping that we are interested in here, though even here there are many different ways to approach the task of creating and using the prototype [Kammer84]. The luxury of being able to create a system only to throw it away has been made possible by the introduction of so-called 'high level' computer languages (Fourth generation languages, and visually based languages such as Visual Basic) that enable working systems to be developed easily and cheaply, though not of a sufficient robustness to be implemented.

Rapid prototyping techniques generally start with a sketchy outline of the desired system. A prototype of the system is made, demonstrated to the user, and his or her comments about its usefulness used as a guideline for its further development. The user might decide that the representation of her requirements is faulty, or that her requirements themselves were erroneous and need to be changed. Once the changes suggested by the user are incorporated into the prototype (this should not take longer than a few days), the new version is shown to the user, and the process is repeated. Once enough iterations have taken place (the decision that the prototype has stabilised is one of the most difficult parts of the method), the prototype is considered to be complete, and can be used as the specification for a 'production' version that can be implemented.

In common with the previous methods, we will ask the questions: is the method scientific; does it successfully distinguish the problem from the solution; and is the presentational notation of any use to us.

The answer to the first question is maybe, but usually not. A prototype can act as a bridge between universal and singular in the same way as a scientific theory: the specification of the prototype consists of universal statements in common with all specifications (the statement of its program is universal), and yet when demonstrated to the user, it becomes anecdotal and exemplary by virtue of its instantiation. In practice, the use of the prototyping approach tends not to be scientific - the anecdotal aspect of the inquiry is used in a fairly unstructured way to reveal requirements for the finished system, not as experiments attempting to refute the universal statement, or theory, embodied in the prototype system, and still less as a means of defining the user domain. Because of this, the approach tends to uncover already well

understood and unambiguous (though probably previously unarticulated) requirements, rather than exploring the limits of system behaviour. The consequences of this are shared by many tools and techniques that 'learn' mechanically (including so-called 'Neural Networks', often hailed as useful 'knowledge' tools for clinicians much as expert systems have been) without providing insight: the majority of the behaviour is as required, but when the system has to react to unexpected combinations of states and operations, its behaviour will be unpredictable. The system created as a result of this process will thus be attractive to the user - 'user-friendly' - but will not display much robustness.

Although rapid prototyping is generally unscientific, it could be used in a more scientific manner assuming the analyst had an accurate conceptual model of the underlying behaviour of the prototype. The technique would still fail by virtue of our second criterion - it does not separate the problem from the solution. Rapid prototyping is an iterative process that refines a potential computerised solution: the requirements that result take the form of the specification of the finished prototype which have to be reverse engineered from it (a far from trivial process). At no stage does the analyst or the user get a deeper understanding of the problem that is being addressed (which in the case of an information system is the organisation being supported, as we have seen). Without any deeper understanding, not only do we miss the 'exceptions' to the general rule, but we also cannot anticipate likely evolution of the system, nor ensure that the system will be able to act as a part of an eventual larger integrated system. Prototype systems tend to be good at satisfying the immediate needs of the users: the absence of any investigation of the underlying processes that are being supported means that ease of modification and maintenance of the system, as well as the facility with which it can be integrated with other systems, will be a fortuitous coincidence rather than a result of the prototyping method.

The processes associated with rapid prototyping are of no great use to us in the initial stages of our analysis - how about the notation? The notation used to record the results of the prototyping-based exploration of users' requirements is the artefact itself, or perhaps a statement of the programme expressed in the programming language used. The statement of the programme takes the form of a mathematical theory. The language is the particular formal system that is used to communicate with the target machine. However, the forms of mathematics expressed as computer languages tend to be obscure and unclear, the syntax confusing, and the semantics disguised (although some computer languages are more 'mathematically pure' than others). An example of this is the popular computer language C++. Even in an environment created to shield the developer from the idiosycracies of the particular machine being used (Microsoft Visual C++ for Windows), we find passages such as follows in the statement of the functional programme module:

```
#include "stdafx.h"
#include "resource.h"
#include "hello.h"
CTheApp NEAR theApp;
CMainWindow::CMainWindow()
{
        LoadAccelTable( "MainAccelTable" );
        Create( NULL, "Hello Foundation Application",
                WS_OVERLAPPEDWINDOW, rectDefault, NULL, "MainMenu" );
}
```

All of this is necessary for the programme to function correctly - none of it casts any light on its essential purpose or requirements.

Similarly, the statement of the programme tends to confuse the ephemeral and incidental with the persistent and essential. This is because the prototype is not simply a statement of the requirements but a

crude working system that needs to be implemented on a physical machine: all computer languages used for conventional rapid prototyping thus reflect this and 'incidental' considerations (such as how to store a particular data type, or how to present information to the user) often obscure the essential logic of the system. In extreme cases, the algorithm of interest occupies a negligible amount of the statement of the programme - for example, C++ for Windows famously requires a programme whose pre-compiled state occupies hundreds of thousands of characters just to print 'Hello World' on the screen. Of course, sometimes it is precisely the incidental that we are interested in, and if we needed to explore the logical consequences of certain data storage strategies, or information presentation protocols, then we would be grateful for (some of) this extra content. At the stage of system development that we are currently at however, this extra verbiage is unnecessary and confusing.

While the above is true for conventional prototyping approaches, there are some which are very close to 'pure' mathematics, avoiding the confusing syntactic encrustations of more common computer languages. Among these are the functional languages such as Standard ML and Haskell, and the specification animation systems such as OBJ3 / 2OBJ and the experimental SUZAN system developed by Surrey University both of which 'execute' mathematical specifications which act as the 'programming language'. The notation for one of these is similar to that chosen to represent the theories developed for this project, and animation would have been used were it not for the unstable state of the animator and the great size of the theory. Animators are generally distinguished from prototyping environments as the presentation of the instantiation of the theory tends to be crude and inefficient, concentrating on the essential functions rather than the user interface or communication with the host machine.

In conclusion, the rapid prototyping method of analysis is not appropriate for the task we are faced with: structuring and understanding the essence of the 'problem domain' which is the clinical directorate. Few rapid prototyping advocates would claim that it was, however. Just as the SSM might be usefully used before a project such as this was defined (in order to produce such a project definition), so rapid prototyping might be used once this project is complete, to ensure that the systems specified do indeed satisfy the needs of the users, to refine the user interface so that it is 'user-friendly', and to act as a vehicle for further experiments which might serve to refute the original theory. The prototyping approach was used with some success to refine the interface for the 'Out-Patient Contract Management System', a system which supported the contracting process, described in Appendix 6.

5.3.5 Others

The above review only covers a small part of the vast range of available systems analysis methods. Many of these are described in the comprehensive review of the field conducted by Matthew Bickerton [Bick92]. This review explores 17 methods in some detail, and a further 22 in a more cursory fashion: all these have been used to help design computer systems, some many hundreds of times. Each of the methods described is unique in one way or another, and more or less powerful at what it does. The degree to which these methods separate the problem and the solution is varied, but all were rejected on the other two criteria. None of these conventional and popular methods explicitly harnesses the power of the scientific paradigm and the notation used by each is semantically weak (Bickerton describes 22 of the presentational notations used by the methods he reviews). The two points are generally linked - if the scientific method is not to be used, then the medium of communication and validation of the theory (or model as many methods call it) will probably be the presentational medium. Because of the difficulty in communicating well defined universal statements (along with their implications) to non-technically minded users, the documentary evidence has to be extremely simple and of great clarity. Some of the popular notations achieve this, but at the expense of accuracy or precision: accuracy when unrigorous and

ambiguous notations are used (such as is the case with DFDs); precision when the complexities of the system being analysed are left out, and a simplified (albeit accurate) representation is provided (such as is the case with ER diagrams).

5.4 Conclusion

In the above chapter we investigated what is meant by analysis, and what common features different analytical techniques exhibit, through the consideration of two very different analytical processes: chemical analysis, specifically of metal salts; and structural analysis, specifically of static structures. It was explained that for each analysis method, a limited set of techniques and processes is used, and the results are expressed in a stylised and structured form (called the presentational notation in the argument above). Both the processes and the notation were considered when the methods chosen for review were investigated.

Methods that might have been used for the project described in this thesis are judged according to how well their associated processes and presentational notations support the scientific method. The degree to which the techniques of the analysis bridge the gap between the universal theory on the one hand and the anecdotal experiment on the other is assessed. If these techniques were taken to be inadequate by this criterion, then the notation was considered to see if it could be salvaged. The criterion for the selection of the method was formal semantic richness - the notation needed to be able to say useful things about the problem formally, and it needed to support rigorous deduction of theorems from the theory.

In addition all methods that are intended to be used to analyse potential information systems should support the clear separation of the problem and solution - in particular by facilitating the description of the domain without any consideration of an eventual computer system.

Three classes of analytical method are considered in this review: data modelling, soft systems methods, and rapid prototyping. For each of these, a typical example of the genre is taken and judged against the three criteria.

The processes used in the data modelling approach, typified in the text by the Structured Systems Analysis and Design Method, do not fit in with the interpretation of the scientific paradigm described in this thesis. The presentational notations display poor semantics: they do not say very much, and consequently deduction from them is extremely limited. As the 'high level' diagrams are refined to a system description, we can see that the problem and the solution are not sufficiently distinguished. Data modelling is rejected as a vehicle for the scientific method - it is claimed that it would be more appropriate to use it as a system engineering tool.

Although they address some important issues, the processes used in soft systems approaches, typified in the text by Checkland's Soft Systems Methodology, do not use the scientific method. The presentational notations, where they are defined, have even less formal semantic content than the notations associated with data modelling. In spite of the observation the SSM effectively supports the separation of problem and solution, it is not considered suitable as a basis of the method that might be used here. The fact that SSM addresses social issues well, and that its perspective on the relation between the 'problem' and 'solution' is philosophically sophisticated mean it is well suited to being used 'upstream' of a project such as this, to define the scope of the project. In this instance this was not necessary, as many of the boundaries of the project (the existence of the clinical directorate structure, the desire for an organisational computer system) had already been decided.

The processes of rapid prototyping, although acting as potential communication channels between the community of computer scientists and that of clinicians, generally do not accord with the scientific method. Even if the approach was modified so that it did so more explicitly, it suffers from the fact that generally no attempt is made to understand the nature of the problem at all, often leading to superficial and fragile information systems. The presentational notation used to record the specification at the end of the process takes the form of the statement of a computer program. For the most part, the syntactic idiosyncrasies of the language used obscure the abstract processes being supported by the prototype. It is argued that just as SSM is more suited to work 'upstream' of the type described in this thesis, so rapid prototyping is more suited to work 'downstream'. Once the underlying structure of the system has been specified, the iterative nature of prototyping and the close and direct involvement of users means that it can be successfully used to refine the system.

Finally, we briefly considered other analytical methods described by Bickerton in his 'Practitioners Handbook of Requirements Engineering Methods'. Although some of these successfully separate the problem and solution, none follows the scientific method, and each is forced to use a clear but semantically poor presentational notation.

It seems that none of the common systems analytical techniques reviewed can help in our desire to follow the scientific method. As the project took the form of a doctoral study the development of a method specific to the project and the problem was not infeasible. Because the hypothesis being tested was specifically that a more explicit and concerted application of the scientific method would result in benefits to the construction of clinical systems, this more thorough course was the one chosen. The method developed is described in the next section.

It might be argued that the wrong selection criteria have been used to assess the methods reviewed. Indeed, some of the issues covered by other analytical methods - those reviewed here and others - are completely ignored in the arguments given above for rejection. Examples of methods and the aspects of the problem they cover particularly well include: Checkland's SSM which addresses the description of social conflicts; Mumford's ETHICS [Mumford86] which tackles job satisfaction; Beer's Viable System Methodology [Beer81] which explores organisational efficiency; Joint Application Development [August91] and Rapid Prototyping which encourage user 'empowerment'. That these other issues are not explicitly covered by the method eventually used does not mean that the methods that address them are not useful, and that we will only consider one 'invented here'. On the contrary, the end product that all of this thought and work is leading to is the design of a successful information system (which is, as we have seen, a difficult proposition): any insight we can gain into the problem can only be beneficial, and the more tools that we use to help in our investigation then the deeper and more varied will be the insights that we achieve.

The purpose of the academic aspect of the project is not to develop a successful computer system - it is rather to test the stated hypothesis and achieve the stated goals. These hinge on the question of whether or not a more explicit use of the scientific method is beneficial to the design of a clinical information system. The use of the other methods described, or at least the adoption of their underlying philosophies, will almost certainly help in the design of information systems - such use will not help us complete the particular task being addressed in this thesis.

Chapter 6: The Method Chosen - An Introduction

6.1 Introduction

This chapter discusses the method actually used in the project. This is necessary if the results are to be understood. However, the aim of the project was to test the worth of the method as well as use it to derive an information system specification. Consequently the method is also discussed in the light of the results in Chapter 13. As in the last chapter the method is described in terms of both the processes and the presentational medium used.

Firstly the presentational medium is explained - this is based around mathematics and symbolic logic. In fact the particular notation used was the Schuman-Pitt notation, a type of model based notation. Model based notations are in their turn a part of a larger family of presentational conventions known as formal methods or formal notations. A formal notation is a structured way of using set theory and discrete mathematics. A justification for the use of the type of notation chosen, and a brief overview of the way that the notation works, is given at each level of selection.

The second section of the chapter describes the processes used to conduct the analysis. These concern the creation of three theories - the domain theory, the information system specification, and the interaction theory which composes the first two - and the use of those theories to engineer an (improved) information system specification.

6.2 The Chosen Method: Presentation

6.2.1 Introduction

In order to appreciate some of the concepts used in the rest of this chapter, and certainly before the results can be understood, we need to consider further the presentational medium that will be used. Some of the arguments presented below are subtle and thus explained at some length - the author makes no apology for this as he feels that clarity is more important than brevity. However, as the nature of set theory and formal notations is not the main subject matter of the project, the ideas introduced here are generally not discussed elsewhere in the thesis.

The idea of the scientific notation is first discussed and the means by which a body of knowledge can be expressed in it described. The use of the appropriate calculus for deriving properties of the system that the theory describes is explained. The choice of discrete mathematics in the form of set theory and first order predicate logic as the basic notation and calculus is justified. A crucial distinction is made in the next section - between theories and models. In short, a theory is a set of rules, and a model of that theory is a 'real thing' that can be considered to obey those rules. The use of a particular variety of discrete mathematics, so called 'formal methods' or more accurately formal notations is explained. Specifically the type of formal method is the model based approach. The philosophy behind this - the representation of systems in terms of an instantaneous state and changes to that state - is presented. A model based formal method can be used to describe systems in terms of rules that are always obeyed by that system - its invariant properties - and those which are obeyed when it is undergoing a particular change - the pre- and postconditions of events. The need for consistency in the theory if any model is to be possible is explained. The way in which set theory was used in the project, taking advantage of its semantic richness but avoiding the use of formal proof procedures is described. The choice of model based formal notations over other forms, and specifically the Schuman-Pitt notation, is justified. Finally, the way in which the so called 'schema calculus' supports the rapid and concise statement of theories of systems is explained.

If the reader feels he or she has a good grasp of formal notations then Sections 6.2.4, 6.2.5, and 6.2.6 may be omitted. If he or she is familiar with the Schuman-Pitt notation in particular then Section 6.2.8 and 6.2.9 may be similarly avoided.

6.2.2 Notations and Calculi: The Choice of Discrete Mathematics

Scientific rigour rests heavily on the use of logical notations with which to describe its theories and relies on calculi to derive the theorems to be tested. We too used a form of mathematics familiar to computer scientists - discrete mathematics in the guise of formal methods - to conduct all stages of the analysis. However a theory is represented (and we might not recognise the mathematics in some forms of presentational notation), the fundamental idea is the same. All theories act as sets of rules that a system[vii] is imagined to obey. Each rule in this basic set is an axiom of the theory: from these axioms we can derive further rules by inference. Mathematicians call the inference mechanisms they use calculi: each branch of science has its own calculi, though they might not refer to it as such. In this parlance, the derived rules are called theorems.

The notations and calculi that are appropriate to our current task - the construction of an information system - are those commonly associated with computer science. Fortunately, there is a generally accepted basic notation and calculus available to us that has historically been used to talk about computer systems. The notation is a set theoretic one, and the calculus the first order predicate logic. Between them these constitute a variety of discrete mathematics. There are versions of discrete mathematics that do not use set theory (for example the mathematics associated with the process description notation CSP), and some that do not use first order predicate logic (for example Imperial College's FOREST which uses modal logic), but by and large the type of mathematics we have chosen has more than enough deductive power and semantic richness to construct the sort of theories of concern to us. The type of notation chosen and the deductive apparatus is a familiar one. An example of a derived theorem is as follows -

given the three relations[viii]

$Daughters: Women \nrightarrow Women$

$Aunts: Women \leftrightarrow Women$

$Sisters: Women \leftrightarrow Women$

and the axioms

$Sisters = Daughters^{-1} \circ Daughters \setminus id[Women]$, and

$Aunts = Sisters \circ Daughters.$

and the rules of logic and set theory, we can deduce that

$Daughters^{-1} \circ Aunts \cap Sisters = \varnothing$

A proof of this theorem is given below in section 6.2.7. The collection of all the theorems that can be derived from the axioms of a theory (including the axioms themselves) is known as the consequence

[vii] system is used in its widest sense here. Theories can describe systems as diverse as the quantum mechanical behaviour of electrons, the properties of the natural numbers, the reproduction rates of geese and foxes, or even the behaviour of human organisations such as a clinical directorate.
[viii] The set-theoretic symbology used here and elsewhere in the thesis is described in Appendix 1.

closure of the theory - it is the consequence closure which lists all the rules that must be obeyed if the theory is correct (However, as Gödel showed, the consequence closure does not describe all the rules that must be obeyed - there is always the chance that some implications of a formal system will not be derivable using a formal calculus).

6.2.3 Theories and Models

While we are considering theories, we ought to address one misconception that is often made concerning the distinction between a theory and a model. A model is:

'... a preliminary solid representation, generally small ... to be followed in construction: something to be copied ... an imitation of something on a smaller scale ...' [Chambers88].

In other words, it is something that takes the form of a quantity of interest. A theory is very different from a model: it does not take the form of the entity in question, it rather lists rules that (we claim) the entity obeys. Thus an entity relationship diagram which purports to describe an organisation would be better described as a theory rather than a model. While a theory and a model are two very different things, we can talk about a model of a theory. A model of a theory is a real thing that obeys the rules set out in the theory. For example if we consider the domain theory, then one possible model of that theory is the Diabetes and Endocrine Directorate (assuming the theory is a good one). We could just as easily produce a computer simulation that has database entities corresponding to the major elements of the theory (its state components), and which obey the rules recorded in the theory. When implemented, such a computer system would be another model of the domain theory. Alternatively, we could consider a computer specification which acts in the same way as a theory. A correctly implemented programme would be a model of that specification. We must bear this distinction in mind when considering the discussion below - the domain theory characterises the clinical directorate, a model of the theory would behave in a similar fashion, and obey the same rules as the clinical directorate (again, assuming the theory is good).

6.2.4 Model Based Notations I: Structural Invariants

As explained above, the notation used is based around set theory, and its associated deductive calculus - symbolic logic. In particular, a branch of set theory was used which was developed for describing a particular class of mathematical systems. These mathematical systems are logically consistent entities which behave in a manner that is both capable of description and which depends on previous behaviours. An example of this sort of mathematical system is an implemented computer programme, and the branch of set theory used to describe them have largely derived from the computer science community. Computer scientists refer to these notations as 'Formal Methods' or 'Formal Notations'.

The type of formal notation employed is sometimes termed 'model based'. A theory presented using such a notation describes rules which a system must obey if it is to be considered a model of the theory. Of course, the system being described is intended to be a 'real' one, be it an organisation, or more conventionally a computer system. How do we know whether these entities are indeed models of the theory - how should we interpret the theory into a model? The model based approach uses a fairly simple philosophy in this respect. The part of the world being described is considered at any instant to be static. Because of this, we can imagine the system being described (instantaneously) by a description of its state - a list of elements that exist, the classifications of those elements, and their relations to each other. If the model is correct (with respect to the theory), its state will be one that is allowed by the theory at that time. Most model based presentational notations use a type of rule to describe the state of the system, called an invariant property. An invariant property, or rule, is one that always holds true over the state of any

model of the theory. This idea may sound confusing, but is in fact quite straightforward, and may be understood more clearly through the use of a (very) simple example.

Let us consider a computer system that will store numbers - perhaps a crude database recording the ages of a group of people (but not their names), the extension numbers of the staff of a department, or recent years producing good claret. We could specify this in the following way -

Dataset: *Set*[$\mathbb{N}^+$]

that is - the dataset is a set of natural numbers (positive integers greater than zero). Let us now assume that the available memory on the computer is very limited, and in fact we are only allowed four entries in the database. We could specify a rule to this end[ix]:

$\#Dataset \leq 4$.

As there are an infinite number of numbers, the possible models of this theory are also infinite in quantity. For each potential model, we have a criterion for judging whether it is indeed a model of the theory as presented - namely we can count the members of the dataset and discard those which have more than 4 (as well as those that contain elements that are not natural numbers greater than zero). We can discuss and describe models directly through the use of the notation of set extension. Here a set is described not by the rules it obeys, but its contents, using the set extension symbols '{' and '}'. Thus the sets described by the set extension expressions

{13, 4645, 9},

{2}, and

{}

(which is the empty set) all obey the rule. The set described by the expression

{56, 1985, 5534, 1, 10, 99354672}

does not as it has six elements. Of course the sets

{93, π} and

{8, 6, "rice pudding"}

are not valid models of the theory either: π is Real but not Natural, and "rice pudding" is not a number.

The above theory is so simple as to be almost useless - for a theory to be valuable it must be able to describe things of many kinds. Thus even a simple database in use will not only store numbers, but also dates, addresses, names, and so on. When describing the clinic we will be interested in patients, doctors, operations, consultations and many other quantities besides. Each of these types is represented by means of a set. A model can be described at any instant by presenting the extension of each of those sets. Each set, which may be simple or complex (such as a relation, function, bag, sequence or other structure), is referred to as a 'state component' of the model. The invariant properties of the theory describe these state

[ix] A glossary of set theoretic terms is to be found in Appendix 1

components and the mutual constraints that act between them. The model of the theory is described by listing the extensions of the state components - that is, enumerating their contents.

6.2.5 Model Based Notations II: Events

We are not only interested in the state of the system being considered - we are also concerned about changes to that state - what are permissible behaviours of models of the theory. Model based approaches again present rules that cover types of behaviour. They do this by listing a number of valid classes of event that can change the state of the model. Each class of event has a name such as 'Create Entry', or 'Clear Dataset', and each is described by listing rules that must hold in addition to the invariants before the event can take place, and those which must hold after the event has been completed[x]. The additional rules that describe the circumstances under which the event is valid are known as the preconditions - those that apply to the state of the model after an event are the postconditions. Again, the ideas here can be better understood through the use of an example.

Suppose the database above was used to record the weight of a new-born child, in ounces, at birth and at monthly intervals thereafter. The weight of a normal baby will increase each month. A simple error prevention mechanism might thus be to only allow weights to be entered that were higher than all the others in the dataset. Suppose we call this operation *'EnterWeight'*, and the new weight to be entered was a quantity x, we might insist on the following rule as a precondition:

$\forall data: Dataset \bullet x \geq data.$

The postcondition to this event would be to ensure that x was now a member of the dataset -

$Dataset_{post\text{-}event} = Dataset_{pre\text{-}event} \cup \{x\}.$

We can see if a model's behaviour is allowed by the theory by inspection. One behaviour might start with the model state

{119, 158, 132}

and *'EnterWeight'* with a new weight of 164. This is allowed as 164 is greater than any of the values currently in the set. The postcondition tells us that the only acceptable new value for the set is

{119, 158, 132, 164}:

any other value for *Dataset* after the event would not be observed in a valid model of the theory. A behaviour where a new weight of 120 was added to the same set would be similarly not be observed in a valid model - 120 is not greater than each of the set's current values.

6.2.6 Model Based Notations III: Consistency

There are certain properties that the theory must exhibit if it is to be able to describe a model. Possibly the most important of these is consistency - it must be possible for a model to obey the rules laid down in the theory. If the theory is mathematically inconsistent then this will not be possible - an inconsistent theory describes nothing[xi]. There are two types of consistency that we are interested in - static and dynamic.

---

[x] Note that in general, model based approaches do not say anything about the state of the system while the event is taking place - other approaches deal with these transient states much more effectively

[xi] This is a simplification (we are forced to make many such by virtue of the use of mathematics): some systems, such as the human mind, can exist in apparently inconsistent states. We can believe in things that might be paradoxical or seemingly inconsistent. Some forms of

Static consistency concerns the invariants alone; dynamic consistency concerns the interaction between the invariants and the pre and postconditions.

Two invariants are inconsistent if there is no model that can obey them. For example, if the *Dataset* theory was developed through the introduction of the invariant

$\#Dataset > 4$

then any model of the *Dataset* state component would have to have fewer than five elements and more than four. There is no set that can satisfy both these criteria, and we can say that the theory is inconsistent and cannot be a valid theory of anything.

In the same way, for a model of a theory to exhibit a behaviour described by an event specification both the precondition and the postcondition must not be inconsistent with any invariant. More subtly, an event must not take place which will leave the model in contravention of one of the invariants. In the database example, with the current invariants and preconditions the following event is not explicitly prohibited. Suppose the model of *Dataset* was

{119, 158, 132, 164}

and the event *EnterWeight* occurred with the value 177 then in the starting state of the behaviour all the invariants and preconditions are satisfied - the cardinality of *Dataset* is less than or equal to four, and the new value is greater than each of the existing ones. However, were the event to take place, then the postcondition tells us that the new state of *Dataset* would be

$Dataset_{pre\text{-}event} \cup \{x\}$.

that is

{119, 158, 132, 164} ∪ {177} = {119, 158, 132, 164, 177}.

But if this were the case, then $\#Dataset = 5$ which contravenes an invariant. We must ensure that all the events allowed by the preconditions and invariants give us legal models after the events. In the database example this would mean the introduction of an additional precondition. One further predicate must be introduced before we can construct the finished precondition - we need to be sure that x is in fact a real number - we need a means of preventing such quantities as π and "rice pudding" from being entered into *Dataset*. The correct precondition to *EnterWeight* is thus

$x \in \mathbb{N}^+ \wedge \forall data: Dataset \bullet x \geq data \wedge \#Dataset < 4$.

6.2.7 Set Theory in Use

One of the great advantages of the combination of formal logic and set theoretic notation represented by the formal method movement is that the formal deduction of one property from another is enabled. Thus the consistency criteria described above can be precisely specified in set theoretic notation and then proven to hold, or to be broken, by the combination of other axioms in the theory. In a similar way, many theorems can be formally derived from a small number of axioms thus easing the search for counter-examples. However, this is not how the notation was used in this project; the deduction of theorems,

logic can represent certain model states and behaviour that would be considered inconsistent and thus impossible using conventional first-order logic and Zermelo-Fraenkel set theory. Such logics are called non-monotonic.

although possible is difficult and longwinded in practice, even where automated support (such as the BTool[xii]) is available. It was felt that the major contribution of the project was in the use of the scientific method to investigate a complex human domain rather than the deduction of interesting theorems. For this reason the majority of the intellectual effort of the project was invested in gaining an understanding of the user's domain rather than exploring the logical implications of the theory.

Nevertheless, the semantic richness and rigour of the theory was not ignored. Properties that were considered to describe the domain under investigation were all expressed as axioms: time and effort was thus not spent deriving theorems, but was rather devoted to ensuring that the (large) set of axioms recorded were mutually consistent. This consistency checking was not done formally (although such formal checking is perfectly possible and examples are given in part 3 of the thesis), but informally and by inspection. The clarity and precision of the notation means that the implications of a theorem are illuminated, and such informal checking is generally sufficient (and where it is not, it is better than the more generally employed alternative where no consistency checking is performed at all).

We can see this clarity if we consider a relatively simple theory and try to express it in both conventional data modelling terms, and also in set theoretic notation. We will take as an example the set of women and the (family) relations between them: specifically aunts, daughters, and sisters. We can express this in LDM terms as is shown in the following diagram (note that *Daughters* is the inverse of a one to many relation - that is, it is many to one):

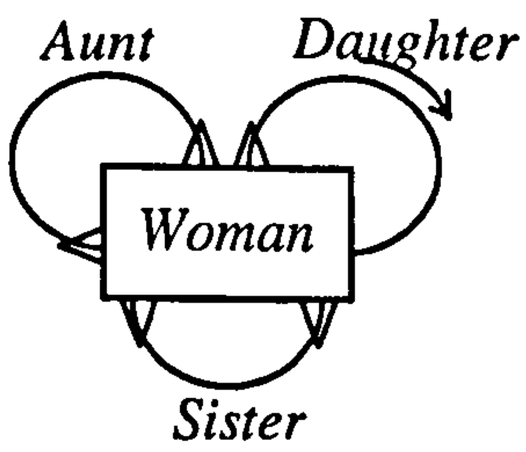


Figure 2-3: LDM of the entity *Woman* and relations over that entity

There are a number of ways in which the relations interact. These are recorded below in textual form as they cannot be expressed in the ER diagram:

- If two *Women* are related to the same *Woman* via *Daughter*, then those two *Women* are also related to each other via the relation *Sister*. The exception to this is if the two *Women* are identical. There are no other elements in the relation *Sister*.

- If one *Woman* is related to another via *Sister*, and a third is related to that second via *Daughter*, then the first and the third are in the relation *Aunt* with each other. There are no other elements in the relation *Aunt*.

We can propose a number of further properties of the theory and, using the facts presented above see if they are logically implied or even at all possible. One of these is as follows:

If a *Woman* is related to another via *Daughter*, and that second is related to a third via *Aunt*, can the first and the third also be in *Sister*?

[xii] A software tool for supporting the discharge of proofs developed by BP's Computer Laboratories

Using the above statements and the ER diagram such a proposition would be very difficult to reason about. Even if we can find the answer a presentation of the argument will be laboured and unclear. If we consider the problem in set theoretic terms, it becomes easier.

We have an element *Woman* taken from some carrier set:

Woman: *Set*[W]

and three relations:

Daughter: *Woman* ↛ *Woman*

Aunt: *Woman* ⇿ *Woman*

Sister: *Woman* ⇿ *Woman*.

The first rule can be translated as the invariant property

$Sister = Daughter^{-1} \circ Daughter \setminus id[Woman]$

and the second as

$Aunt = Sister \circ Daughter$.

The proposition in question can be rephrased as:

Can pairs in $Daughter^{-1} \circ Aunt$ also be in *Sister*?

If we can show that $Daughter^{-1} \circ Aunt \cap Sister = \varnothing$ for all *Aunt*, *Sister*, *Daughter*, *Woman* then the proposition will be false. This is what we will do below.

Using equalities which are the invariants, we can say that

$Daughter^{-1} \circ Aunt \cap Sister = Daughter^{-1} \circ Sister \circ Daughter \cap Sister = Daughter^{-1} \circ (Daughter^{-1} \circ Daughter \setminus id[Woman]) \circ Daughter \cap Daughter^{-1} \circ Daughter \setminus id[Woman]$.

But from the observation that backward relational join '$\circ$' is a disjoint operation, we can say that

$Daughter^{-1} \circ (Daughter^{-1} \circ Daughter \setminus id[Woman]) \circ Daughter =$
$(Daughter^{-1} \circ (Daughter^{-1} \circ Daughter) \setminus Daughter^{-1} \circ id[Woman]) \circ Daughter =$
$Daughter^{-1} \circ Daughter^{-1} \circ Daughter \circ Daughter \setminus Daughter^{-1} \circ id[Woman] \circ Daughter =$
$Daughter^{-1} \circ Daughter^{-1} \circ Daughter \circ Daughter \setminus (Daughter^{-1} \circ id[Dom(Daughter)] \circ Daughter \cup B)$

Where $B = Daughter^{-1} \circ id[Woman \setminus Dom(Daughter)] \circ Daughter$.

But a basic rule of set theory tells us that $A \circ Cod(A) = A$ for any relation A. We thus know that

$Daughter^{-1} \circ Daughter^{-1} \circ Daughter \circ Daughter \setminus (Daughter^{-1} \circ id[Woman] \circ Daughter \cup B) =$
$Daughter^{-1} \circ Daughter^{-1} \circ Daughter \circ Daughter \setminus (Daughter^{-1} \circ Daughter \cup B) =$
$X \setminus Daughter^{-1} \circ Daughter$

where $X = Daughter^{-1} \circ Daughter^{-1} \circ Daughter \circ Daughter \setminus B$.

The expression

$Daughter^{-1} \circ Aunt \cap Sister$

can thus be written

$X \setminus Daughter^{-1} \circ Daughter \cap Daughter^{-1} \circ Daughter \setminus Y =$
$X \setminus Sister \cap Sister \setminus Y$

where $Y = id[Woman]$.

It is clear that this intersection is the null set - whatever X is, the intersection of X excluding *Sister* with a subset of *Sister* is empty. This might be considered to be a trivial example of the power of set theory but it suffices to demonstrate the rigour and flexibility of the notation and its calculus. From the axioms given, we have been able to clearly show that the daughter of a woman's aunt can never be her sister. If we find a refutative example (and this is not incredible), then either the axioms are incorrect, or the proof given above has been discharged incorrectly.

Although the terms used in the above argument are expressed in formal terms, the argument itself is not: English phrases such as 'We know that', 'But', and 'Where' are used as conjunctions in the argument without giving them any formal semantics. A formal proof would not use such terms, and the conjunctions in the stages of the argument would themselves have formal definitions. Such arguments are often called 'proof trees' and , because of the formality of the semantics of the argument, can be generated (semi-) automatically. Much of the reasoning presented in the next part is even less formal than the one we have just seen: however, it all exploits set theory to a greater or lesser extent thus adding to the force and clarity of the arguments used.

The next two sections will discuss the particular brand of formal notation used in the project: the Schuman-Pitt formal notation. The reasons for its choice and its unique features will be considered.

6.2.8 The Schuman Pitt Notation I: Why Was it Chosen?

Having decided that the appropriate mathematics to use is the discrete variety in the form of set theory and first order predicate logic, and moreover that the conventions associated with formal methods are useful we are still faced with a bewildering array of notations and approaches. Which formal method should we use and why?

Cohen [Cohen84], [Cohen91] divides formal methods into three types (though he also considers structured methods such as data modelling as having a minimal degree of formality and thus semantics): the algebraic, the process based and the state or model based. Any formal notation can be used to describe the behaviour of any formal system, such as a computer, but some classes of system are more susceptible to description by certain types of notation than others.

Algebraic notations were designed to describe and reason about abstract data types (ADTs). Examples of such notations include CLEAR, OBJ, and Miranda (which is a functional programming language). An abstract data type describes a class of data that shares structural and behavioural properties. The algebraic notation specifies a valid 'language' and 'grammar' that can be used to describe the value of a particular ADT. The language is composed of terms which are instances of the ADTs, and those which are operators

which take those instances and return others of the same or different ADTs. Behaviour as such is not represented, nor state - only the nature of valid operations on instances of ADTs. Because of this, notations of this kind are useful for describing operations on data types supported by a given computer language (such as the behaviour of the ADT 'set', 'list', or 'number' in a particular language), or for specifying compilers and interpreters which transform data of one type (programme notation) into another (machine readable code). Describing information systems or organisations in terms of ADTs and the operations on them would be possible but tedious and result in a theory that was difficult (for the analyst) to interpret into the domain.

Process based notations are used to describe behaviours of systems that are composed of interacting parts. Examples of process based notations are CSP, CCS, and Occam (which is the programming language for computer systems built using the Transputer parallel processor). A particular strength of these approaches is that they enable the composition of and reasoning over actions and classes of action. Specifically they help us to investigate the effects of concurrent processes in the same system. A computer system can be thought of as just such a confluence of processes - the actions it engages in depend on the actions which are interfering with it, past actions it has engaged in and actions it is currently attempting to complete. Note that the factors determining which actions the process will engage in are other actions, not the state of the system: in fact the state of the system is not represented at all in most process based notations. For this reason, process based notations are particularly useful for describing a system without much apparent internal state, but where the interference between processes is important. Examples of such systems include a telephone network (where the behaviours of telephones and the central exchange will interact in subtle and complex ways) or an operating system (where the behaviours of the different aspects of the system - storage, display, processor, keyboard - will interfere with each other). A solely process oriented method is not suitable for describing systems which can be best thought of as having a complex internal state such as an information system or an organisation.

Notations which were solely algebraic, and those which were solely process based were deemed unsuitable for our purposes: namely the investigation and description of an organisation and the information system that is to support it. It seemed that the easiest way to represent an organisation was to adopt the underlying philosophy of the model based methods: representation via models of the theory that possess instantaneous state, and where behaviour is dependent on that state. This was also considered to be an appropriate notation for the specification of an information system which would be implemented as a relational database.

Having decided that the model based notations are most suitable for the tasks embarked on, there are still several possible methods to choose from. Two very popular ones are VDM [Jones90] and Z [Spivey89].

The Viennese Development Method was developed by IBM in the 1970s: it is not only one of the earliest formal notations, but also one of the most widespread. Although it is a popular notation, has been used in many commercial projects, and there is much literature related to it, more recent developments feature improvements over the technique. The most obvious drawback with VDM is the paucity of the structuring conventions: theories have to be specified 'all at once': complexity cannot be introduced incrementally - in fact the notation itself is very similar to 'raw' set theory.

Z, developed at the Oxford University Computing Laboratory in the early 1980s (from original work by J R Abrial), is another notation that is popular, is extensively supported by literature, and has been used in commercial projects (for example in the widely reported re-development of the IBM product CICS [Hayes93]). The notation features more structuring than VDM and introduces a 'schema calculus' which

enables specifications to be combined and refined incrementally. Although this is an improvement over VDM, Z does not readily support composition of several systems which mutually interfere in the way that the process oriented notations do. Support for such composition would be very useful in the process of analysis for two reasons. Firstly we want to investigate different aspects of the organisation in isolation and see how they interact, thus reducing the complexity of the work through the effective 'separation of concerns' (this concept is discussed more fully in Section 6.3.3). Secondly we need to compose two different theories together - the domain theory and the information system specification - to form the interaction theory.

Composition of processes and systems is supported in a model based framework by the Schuman-Pitt notation. This is from the same stable as the more common Z (they are both developments of earlier pioneering work conducted by J R Abrial) and was developed by Steve Schuman and David Pitt at the universities of East Anglia and Surrey. Using this method we can talk about processes and compose those processes together (as we can with CSP and CCS) so as to analyse possible behaviours (expressed as 'traces' of events) of the composite system. The advantage of the notation over the more conventional formal process description languages is that it is model based and thus enables the expression of theorems that describe and constrain the possible instantaneous states of the system. It does this through its own schema calculus which describes the means of composing two theories and the resulting semantics of the composition. One remarkable feature of the notation's semantics which helps class composition is the support for indeterminate post-conditions. The Z notation insists that the state of an object following an event is completely specified. This is a problem when composing classes as great care must be taken that postconditions from composed operations are not mutually inconsistent. In designing a theory or specification in Z, component classes often have to be re-engineered to reflect a constraint that the composite class imposes. In other words, specifying in Z benefits enormously from the wisdom of hindsight. The Schuman-Pitt notation on the other hand is more flexible when it comes to composition. Instead of insisting on deterministic specification of postconditions, more indeterminate predicates can be used. In short, the specifier is able to define what the essential changes to the state of the system will be after the operation, and the semantics of the notation are such that only those changes that are necessary to ensure that the invariants are not contravened are implied. By only describing the essential changes, composition is possible without the necessity of re-engineering the component classes. This is described further in the next section. For an in depth review of the concept of the weakest postcondition, the reader is referred to 'the Rest Stays the Same' [Schuman94] by Steve Schuman and Dave Pitt. It should be noted here that another benefit of the semantics is that the specifier needs to concentrate on making the strongest invariants possible, and the weakest postconditions. This is in general a good design strategy anyway. It enables us to tightly specify the topology of the system's state space which can not only reveal valuable insights but is substantially easier to reason over than the state space represented by the sum of all possible behavioural traces.

Before we leave the subject of class composition, a significant difference between the Schuman-Pitt notation and other Object-Oriented (OO) specification notations should be commented on. Whereas a common feature of many OO notations is the representation of mutual constraints imposed by two classes on each other through the passing of messages between them, the Schuman-Pitt notation directly represents these as direct conjunction and specification of new invariants over the conjoined class. This again means that we can directly and tightly define the topology of the system's state space rather than having to infer it by considering sequences of messages and their effect.

Because of these advantages (and the willingness of one of the originators of the notation, Steve Schuman, to work closely with the author) the Schuman-Pitt notation was the one chosen as the means of

expressing and presenting all the theories. The next sub-section gives a very brief overview of the Schuman-Pitt notation, and describes some of its unique features.

6.2.9 The Schuman Pitt Notation II: How Does it Work?

As explained earlier, the basic notation used is set theory (specifically the strongly typed variety defined by Zermelo and Fraenkel in the early years of this century [Hayden68]) and the associated calculus first order predicate logic. The Schuman-Pitt notation provides a set of structuring conventions so that we can easily describe the behaviour of complex systems.

A theory is created in the notation out of syntactic and semantic units. The syntactic, textual, unit is the schema. A schema looks somewhat like a table with set theoretic predicates as its contents. The semantic unit is the class. The Schuman-Pitt formal language is 'object-oriented' which is one of the reasons why it is so powerful - objects are not described directly, but the classes which act as the containers of the object are. Using the various schemata, a theory of the behaviour and states of objects which are instances of the named classes can be constructed. One of the important concepts of the object oriented approach to system description is the notion of inheritance. Thus a class whose instances are objects with simple states and behaviours can be refined to create a class whose instantiated objects are more complex. The development of the theory of one class into a theory of another class is made possible by the 'schema calculus' which describes how a schematically described class can be refined, and defines the semantics of such refinement. The schema calculus supports multiple inheritance - in other words a class can be composed of two or more others in which case its instantiated objects will in some sense exhibit properties of all the inherited classes. How the schemata describe the states and behaviours of objects, and how such schemata can be combined to form new theories is explained briefly below.

There are two types of schemata: state schemata and operation schemata. A class is described by one state schema and a number of operation schemata all of which share the same tabular form with up to three areas where descriptions can be entered as follows:

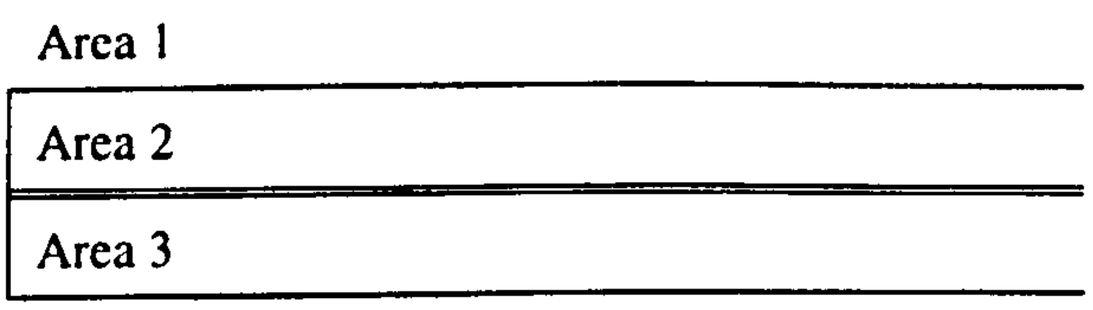


Figure 2-4: Tabular format of state and operation schemata.

Note the three areas where predicates and other set theoretic expressions can be entered: Area 1 (at the head of the schema), Area 2 (above the line), and Area 3 (below the line).

The state schema of a class describes the rules limiting the possible states of an object of the class. The name of the class of objects currently being described by the theory is written in area 1 at the head of the schema. In the body of the schema, above the line, type declarations and invariants are presented. The type declaration defines the most basic nature of the state components of the objects in the class in terms of more primitive state components (the most primitive of which are the carrier sets, the basic types in the theory). The invariants are set theoretic predicates which describe allowable states of the state components. Below the line, in area 3, the initialisation pseudo-operation is written: what value objects in

the class start with are recorded here. Using the example of the database theory introduced earlier we can illustrate the way in which the state schema of the Schuman-Pitt notation describes the state components and invariants of objects in a class.

BasicDatabase

Dataset: *Set*[$\mathbb{N}^+$] (Type Declaration)
$\#Dataset \leq 5$ (Invariant Predicate)
$Dataset' = \varnothing$ (Initial Value)

The 'dash', or 'prime' character following the name of the state component under the line indicates that this is its state after initialisation.

The operation schema describes a class of events that the objects being defined can engage in (In this thesis, events in the model's behaviour are called operations). At the head of the schema we write the name of the operation, and any parameters that the operation takes. Thus in the database example, a name of one of the operations is *EnterWeight* and the parameter is the weight to be entered to which we assign a (dummy) label thus: *EnterWeight*(x). In area 2 the parameters are described in terms of the state components (be those primitive or specified) of the class - in our example the parameter, x, must be a natural number so we say:

$x: \mathbb{N}^+$.

Once the parameters have been described we can write the precondition to the operation - this is also done above the line in the second area. Finally, below the line we write the postcondition - the prime character is again used to distinguish the value of a state component after an operation from its value before. The example for EnterWeight would thus be:

BasicDatabase.EnterWeight(x)

$x: \mathbb{N}^+$
$\forall data: Dataset \bullet x \geq data \wedge \#Dataset < 4$
$Dataset' = Dataset \cup \{x\}$

.

A class theory can be refined, composed with another class theory, or composed and refined in order to create a theory of a new class. The semantics of this composition is given as part of the schema calculus and is precisely defined. In general we can give the form of a typical state schema as follows:

Class1

$X_1: PT^{c1}{}_1; X_2: PT^{c1}{}_2; \ldots ; X_{nt}: PT^{c1}{}_{nt}$
$P_1(X_1, \ldots, X_{nt}); \ldots ; P_{np}(X_1, \ldots, X_{nt})$
$I_1(X_1', \ldots, X_{nt}'); \ldots ; I_{ni}(X_1', \ldots, X_{nt}')$

.

In the above schema, PT_i is a primitive type, so the first line in area 1 is declaring types (The semi-colon enables two type declarations or predicates to be written on the same line - semantically, two predicates on adjacent lines, or on the same line but separated by a semi-colon can be thought of as being joined by the $\wedge$ conjunction). $P_i(X_1, ..., X_{nt})$ is the ith invariant predicate over the state components $X_1, ..., X_{nt}$. Similarly $I_i(X_1', ..., X_{nt}')$ is the ith initialisation predicate over the same state components dashed. Suppose we had defined the invariants of another class as follows:

Class2

$Y_1: PT^{c2}{}_1; ... ; Y_{mt}: PT^{c2}{}_{mt}$

$Q_1(Y_1, ..., Y_{mt}); ... ; Q_{mp}(Y_1, ..., Y_{mt})$

$J_1(Y_1', ..., Y_{mt}'); ... ; J_{mi}(Y_1', ..., Y_{mt}')$

.

We can compose the state schemas of these classes together to give the state schema of a third as follows:

Class3

Class1, ***Class2***

$Z_1: PT^{c3}{}_1; ... ; Z_{lt}: PT^{c3}{}_{lt}$

$R_1(X_1, ..., X_{nt}, Y_1, ..., Y_{mt}, Z_1, ..., Z_{lt}); ... ; R_{lp}(X_1, ..., X_{nt}, Y_1, ..., Y_{mt}, Z_1, ..., Z_{lt})$

$K_1(X_1', ..., X_{nt}', Y_1', ..., Y_{mt}', Z_1', ..., Z_{lt}'); ... ; K_{li}(X_1', ..., X_{nt}', Y_1', ..., Y_{mt}', Z_1', ..., Z_{lt}')$

.

The schema calculus tells us that the meaning of this schema is given by replacing the word Class1 with the type declarations and invariants from that class (Class1) above the line, and the initialisation predicates below. The equivalent (and normal form) expanded class schema would thus look like this:

Class3

$X_1: PT^{c1}{}_1; ... ; X_{nt}: PT^{c1}{}_{nt}$ .....*Type Declarations*

$Y_1: PT^{c2}{}_1; ... ; Y_{mt}: PT^{c2}{}_{mt}$

$Z_1: PT^{c3}{}_1; ... ; Z_{lt}: PT^{c3}{}_{lt}$

$P_1(X_1, ..., X_{nt}); ... ; P_{np}(X_1, ..., X_{nt})$ .....*Invariant Predicates*

$Q_1(Y_1, ..., Y_{mt}); ... ; Q_{mp}(Y_1, ..., Y_{mt})$

$R_1(X_1, ..., X_{nt}, Y_1, ..., Y_{mt}, Z_1, ..., Z_{lt}); ... ; R_{lp}(X_1, ..., X_{nt}, Y_1, ..., Y_{mt}, Z_1, ..., Z_{lt})$

$I_1(X_1', ..., X_{nt}'); ... ; I_{ni}(X_1', ..., X_{nt}')$ .....*Initial Values*

$J_1(Y_1', ..., Y_{mt}'); ... ; J_{mi}(Y_1', ..., Y_{mt}')$

$K_1(X_1', ..., X_{nt}', Y_1', ..., Y_{mt}', Z_1', ..., Z_{lt}'); ... ; K_{li}(X_1', ..., X_{nt}', Y_1', ..., Y_{mt}', Z_1', ..., Z_{lt}')$

.

It is because the type declarations and the predicates are inherited from the composed classes that we can create invariant predicates in the new theory that refer to state components defined and constrained in

previous theories. The necessity of consistency between invariants in a class thus does not apply solely to the invariants specified in that class's state schema, but to the set of invariants in the expanded schema. It should be noted here that the primitive types that are used in the type declarations in the composed class might be constructed out of the declared types in the earlier schemata. If this is the case then the type declarations in the unexpanded schema of the composite class must be rewritten as declarations which do not use previously defined types (as opposed to given types, such as the carrier sets) and appropriate invariants which constrain the now relaxed types. For details of this and other subtleties of the schema calculus as it applies to the composition and refinement of state schemata the reader is referred to the original literature [Schuman90].

Operation schemata are composed together in a similar way - the expanded schema taking all the preconditions and postconditions from the composite schemata as well as the unexpanded one. Of course we need to specify not only the classes being composed, but also the operations within those classes. One important aspect of the notation that was commented on above is the idea of minimal specification of postconditions. This point merits further consideration.

Suppose in our database example we had defined an operation, *RemoveWeight*, to remove the lowest value element from the state component *Dataset* (This would have the effect of changing the starting point of the weight measurements from birth to one month, then two months and so on). A reasonable definition for this operation might be:

BasicDatabase.RemoveWeight

$$Dataset \neq \emptyset$$

$$Dataset' = Dataset \setminus \{x: Dataset \mid \forall data: Dataset \bullet x \leq data\}$$

Which has the effect of removing the lowest value of *Dataset*.

One possible refinement of the dataset would be to introduce a new operation on a subsequent refined schema which kept the cardinality of *Dataset* the same, but changed one of its members - thus the updating of the database could be done in one operation rather than two. This could be specified by creating a new operation (in a derived class) which invoked both the EnterWeight and RemoveWeight operations from the previous class thus:

LessBasicDatabase.ChangeWeight(x)

BasicDatabase.EnterWeight(x)

BasicDatabase.RemoveWeight

There is no 'below the line' section here because we need to specify no new postconditions. With the existing postconditions for EnterWeight and RemoveWeight, however, there is a problem. As the postconditions are inherited and conjoined the postcondition of the new operation would be:

$Dataset' = Dataset \cup \{x\} \wedge Dataset' = Dataset \setminus \{x: Dataset \mid \forall data: Dataset \bullet x \leq data\}$. With the specified preconditions this is inconsistent: from the first postcondition we have

$Dataset' \subset Dataset$

(because we know that $x \notin Dataset$)

but the second tells us that

$Dataset \subset Dataset'$

(as we know that $Dataset \neq \varnothing$).

These two propositions cannot both be true. The Schuman-Pitt notation allows us to be less prescriptive about the postconditions but still say what we want to. For example, instead of writing

$Dataset' = Dataset \cup \{x\}$

as the postcondition for the first operation, we can say

$x \in Dataset'$

and for the second operation (that removed the lowest member)

$\{x: Dataset \mid \forall data: Dataset \bullet x \leq data\} \cap Dataset' = \varnothing$.

These two propositions are not inconsistent. The semantics of the notation mean that we can deduce the exact state following an operation from these seemingly less deterministic postconditions: the specified change is accommodated but where possible, the rest of the state of the object described stays the same. Thus in the case of the earlier postcondition, *Dataset* has x as a member after the operation, but is otherwise unchanged. Exactly what 'the rest' which remains unchanged is rigorously specified by the notation's semantics. In this area these are subtle and somewhat involved, and they will not be explained further here. In use however, the meaning of the operation schemas is informally clear without taking recourse to the formal details of the language - the idea of 'minimum change' is intuitive if difficult (though certainly possible) to define rigorously.

While considering consistency, we can briefly discuss the proofs we are obliged to discharge to demonstrate that a class is consistent. The Schuman-Pitt notation insists on a predetermined number of proof obligations for each class. For each state schema, we should discharge two proofs - one to show that a valid model of the class exists, and the other to show that the initial state of the class is a valid one. Clearly these two proofs can be discharged with the same argument that demonstrates that all possible initial states of the class are valid. Each operation is similarly associated with two proof obligations - the first shows that a state of a model of the class that satisfies the invariants and the preconditions of the operation exists, and the second that for all such initial states, a state exists that satisfies the postconditions and invariants (in their dashed forms). Thus for each class, $2n + 2$ (where n is the number of operations in the class) proof obligations must be discharged if we want to guarantee consistency. Although formal proofs were in general not discharged in this project, the fact that we know exactly how many there are, and exactly what form these take, ease the task of informal consistency checking by inspection. Simple proofs for the consistency of one class are conducted in Section 9.2 of the thesis.

One final point will be made here before we proceed to the next section of the chapter. Although the notation is heavily 'object oriented', not all of the features of this 'paradigm' were used in the theories created by the project. Notably, the specification of separate objects that communicate and interfere with

each other was avoided. Thus a model of the complete theory can be thought of as being one object that has instantiated all the properties of all of the classes described in the theory. The fact that the organisation is described by defining one highly complex object as opposed to a network of simpler interfering ones renders the semantics of the theory simpler and thus hopefully easier to understand. A reworking of the ideas making more use of this aspect of the notation might be a fruitful exercise however, as is pointed out in section 14.7.

The next section describes the processes used in the analysis (as opposed to the presentation of its results).

6.3 The Method: Process

6.3.1 Introduction

This section will talk about the procedures and processes used in the analysis. Firstly we will talk about the need for three separate theories - the domain theory, the information specification and the interaction theory that composes these two together. The domain theory describes the part of the organisation under investigation; the information system specification describes an existing or proposed information system as a basis from which we engineer future changes; the interaction theory composes the two together so that the adequacy of the interpretation of the information system into the domain can be assessed. The method used for deriving the domain theory is presented briefly. This revolved around the use of singular statements in the form of anecdotes elicited during interviews with clinicians to refute and thus refine universal theories of the organisation. The information system specification describes either a current information system that needs to be improved and integrated more successfully with the organisation, or a proposed information system that will address a particular need. The interaction theory composes the domain theory and information system theory together to create a third metaphysical theory which attempts to show how the latter will be perceived in use as a representation of the former. Finally, the last three sub-sections consider how the combination of these three theories can be used to engineer a better information system.

6.3.2 The Need for Three Theories

The previous chapter introduced one of the most fundamental assumptions on which this work is based: namely that when in use, an information system is interpreted into the world by its users. In other words, an implemented information system is understood to be a representation of a component of the world as the user perceives it. Furthermore, if the information system cannot be interpreted (or can only be partly interpreted) as an aspect of the world, then the users will display dissatisfaction with the system. We cannot access either the user's perception of the world (called the domain) or her interpretation of the information system directly. We investigate this indirectly, however, by saying (more or less accurate) things about the domain and seeing whether an appropriate interpretation is possible for the designed information system, assuming that the things we said about the domain were correct. This is illustrated in figure 2-5 below.

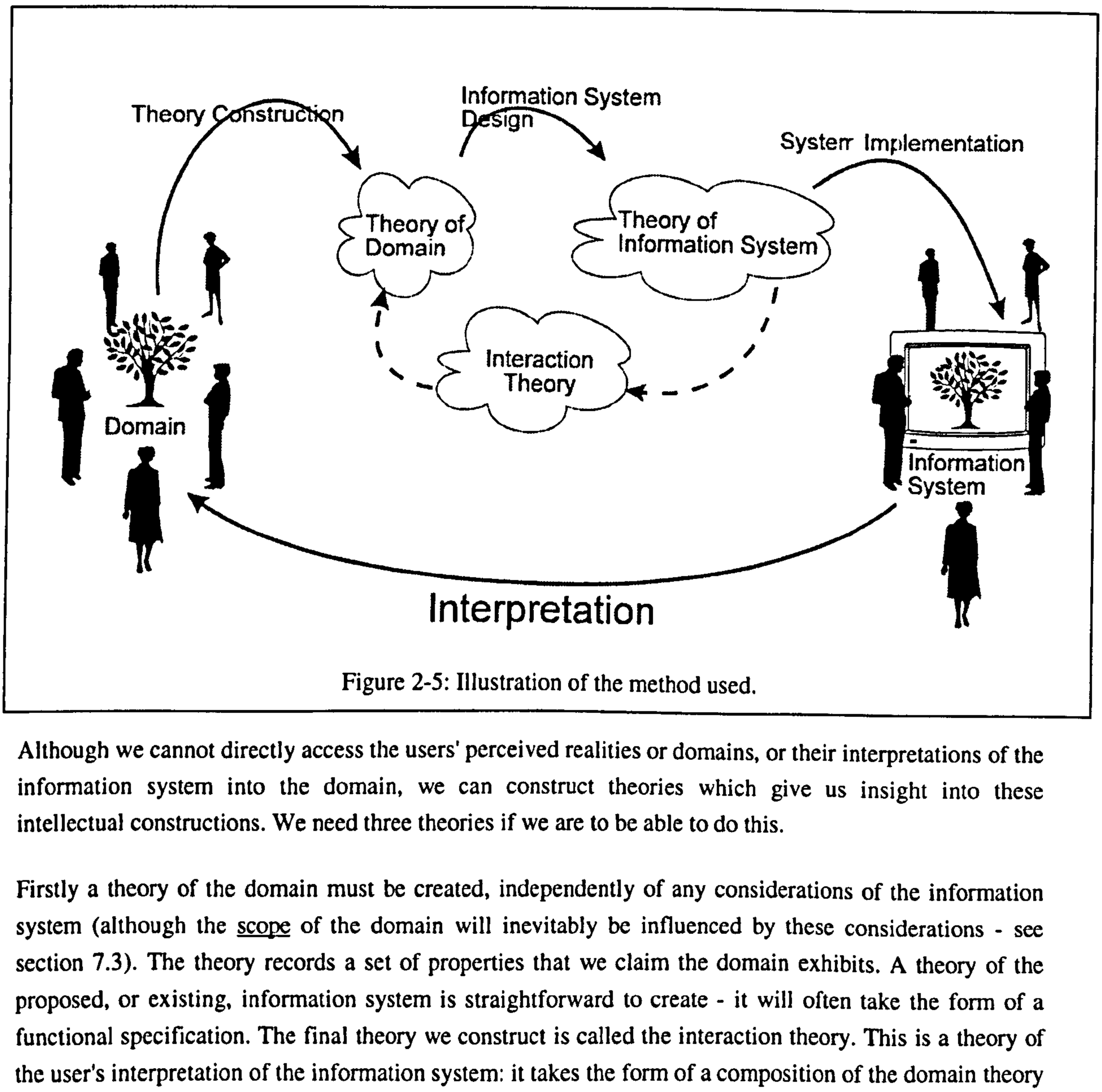


Figure 2-5: Illustration of the method used.

Although we cannot directly access the users' perceived realities or domains, or their interpretations of the information system into the domain, we can construct theories which give us insight into these intellectual constructions. We need three theories if we are to be able to do this.

Firstly a theory of the domain must be created, independently of any considerations of the information system (although the scope of the domain will inevitably be influenced by these considerations - see section 7.3). The theory records a set of properties that we claim the domain exhibits. A theory of the proposed, or existing, information system is straightforward to create - it will often take the form of a functional specification. The final theory we construct is called the interaction theory. This is a theory of the user's interpretation of the information system: it takes the form of a composition of the domain theory and the information system theory. If the interaction theory is well constructed, it can reveal the extent to which an information system that is a correct implementation of the specification (that is, the information system theory) can be interpreted into the domain. This may reveal a number of shortcomings in the information system. For example: an entity in the information system might exhibit an insufficient range of behaviours when interpreted into the domain; the entity might exhibit behaviours that are never observed in the domain; the entity might exist in insufficient states thus not representing the required

extent of the domain; the entity might exist in states that represent states of the domain that are never observed; or the entity might exhibit a number of these inaccuracies. Thus through inspection of the interaction theory, we can understand how the information system will be interpreted and determine where that interpretation is problematic.

Understanding the world is the role of science - acting to change it is the role of engineering. We can engineer better systems specifications once we have gained (through use of the scientific method) a better understanding of the parts of the world we are interested in (namely the domain, the information system, and the interpretation of the information system into the domain by the users) and expressed it in the form of the three theories. By inspection of the interaction theory we can see where interpretational weaknesses of the information system lie, and can thus take action to overcome them.

This engineering step is not mechanical, and there is no 'correct' way to complete it. It is often not desirable to have the computer system act as a perfect representation of the domain. In fact, it is argued elsewhere (Section 12.4) that in general we should strive for simplicity and minimalism in implementation. There are, however, guidelines that can be used - some errors of representation are more serious than others. By using the interaction theory, we are confronted with the implications of the design decisions that we make. In this way the method advocated here enables the design of the information system to be completed in an informed manner - neither we as designers, nor the users (assuming they are consulted in the engineering process) should be presented with any unpleasant surprises upon implementation.

It should be remarked here that the domain theory is not the same as the requirements of an information system. Even if we accept our assumption concerning the interpretation of the Information System, we have not in our analysis identified for which aspects of the domain support is required. It is unlikely that the entirety of the domain analysed should be given computer assistance, and even if it should, computerisation of different sub-domains will have differing priorities. The statement of the areas of the domain that require most urgent support is a different sort of statement of 'requirement', equally important, but not addressed by the method discussed in this thesis. In practice, the choice of system implementation ordering will in all probability be made by the commissioner of the analysis. The existence of the domain theory is crucial here so that the meaning of a requirement for a 'booking system', or 'an internal referral system' is clear, and the implications of implementing support for those sub-domains alone is apparent.

If the engineering decisions described in the previous paragraph are to be made wisely, we must ensure that our three theories are as accurate as we can make them. If the information system being investigated already exists then the information system theory must be derived from it - a classic case of 'reverse engineering'. If it does not, then the information system will be an implementation of the specification, and accuracy becomes an irrelevant concept (it will be important to provide an accurate implementation, but we are not concerned with that here). Similarly, if the information system is being designed, then the interaction theory will be almost impossible to verify (though this does not prevent it from being wrong: see Section 13.4) - even if the information system exists, the interaction theory will be very difficult to test because of the tenuous and indirect nature of the domain being described (the interpretation of a symbolic structure into a perceived reality). The essential problem is that we cannot predict how the information system will be used any more than we can predict how any tool will be used. We cannot demand that the user of an information system interprets the entity named 'patients' as the real-world concept patients and not hospitals, doctors, or drug regimes any more than we can demand that the user of a screw-driver uses the tool to tighten screws rather than open tins of paint, bang in nails, or stab passers-by. This does not

relieve us of the obligation to create as realistic and honest an interaction theory as we can - merely that the scientific method is of limited use in so doing. In practice, the interpretation of entities and operations in the information system are generally fairly obvious, and introspection is usually sufficient to create a good interaction theory. The technique of constructing the interaction theory is examined in greater detail in Section 11.3. The domain theory, however, can be tested extensively and according to the scientific method - this must be done if the information system is to have a chance of benefiting from the approach described here. The quality of the decisions made concerning the engineering of the system specification depends on the accuracy of the various theories required: it is beholden on us to ensure that these are as 'correct' as we can reasonably make them.

The following three sections describe in more detail how the three theories are constructed, and how they are used to engineer an information system specification. We will presently consider the construction of the first of the three theories - domain theory - but first we will make some important observations about theories in general, and the notation employed.

6.3.3 The Domain Theory

The method used to construct the information system depends crucially on the central assumption concerning interpretation of the information system: in order for it to deliver an effective system, a sound understanding of the domain is required. This understanding is expressed as the domain theory - it is this theory which acts as the starting point for the engineering of the system specification. If we get the domain theory 'wrong' then the whole process is rendered invalid. Conversely, the better our domain theory, the more likely we are to identify potential interpretational shortcomings in the eventual information system. However, not only is the domain theory the initial and thus the most important of the theories to get 'right', it is also the most difficult. The attempt to understand any person's world view is fraught with difficulties of a philosophical as well as a practical nature: that the world view concerns a subject area as complex as the delivery of health care increases still further the problems we face.

From the above preamble, it should come as no surprise that it is the construction of the domain theory that takes the greatest time, and it is here that the greatest degree of intellectual effort should be invested. In the event, the development of the domain theory took eighteen months - half the time allotted to the entire project (excluding 'writing up'). The observation that this part of the project would be the most important and demanding meant that it was here that the scientific method was followed most scrupulously. The manner in which the scientific method was used is described in the next few paragraphs (a deeper and more detailed discussion is presented in Section 13.3).

The scientific method relies on the development of ever bolder (or more falsifiable) theories and their subsequent refutation and reconstruction. In order to start the process, an initial theory is needed. This is a first guess or best estimate, derived inductively from a combination of observations, discussions and the analyst's prejudices. The scientific method says nothing about this process which is a fundamentally creative one. In the same way it does not present a way of creating a new theory when an existing one is refuted - it merely sets out criteria for judging the merit of a given theory. A discussion of the creative process lies more in the realm of psychology than systems analysis and will not be discussed further here. Suffice it to say that it is clearly one of the most fundamental, if ineffable, components of any difficult intellectual endeavour - analysis is no exception and the use of mathematics to help judge its results should in no way be taken as an attempt to 'automate' this most human of activities. For interesting and illuminating discussions of the nature of creativity, the reader is referred to, for example [Koestler89].

In the case of the analysis reported here the initial theory was constructed after the analyst had explored the domain in an informal manner, by means of observation of the clinic in operation and discussion with workers in the directorate. Once the initial theory was defined and stated, the cyclical theory development process could be embarked on.

As was explained earlier, the theory consists of axioms and rules of deduction with which theorems can be derived. If any theorem is seen to be incorrect, then the axioms must be false (or possibly the rules of mathematics, though this is a lot less likely!), and as a result the theory itself is refuted. The rules that govern our lives tend to be of the 'thou shalt not' rather than 'thou shalt' variety (at least in English law): society decides the limits of acceptable behaviour within which its citizens may act as they wish. The rules which together form the theory of the domain are no different - they describe the limits of acceptable behaviour of the system. The difference between these two bodies of law lies in the appropriate reaction to the breaking of a rule. In the case of a citizen breaking the rules of the society she lives in, that person is considered to have transgressed and will recieve some form of sanction. In the case of the system exhibiting a behaviour that is forbidden by the theory that purports to describe it, the theory is considered to be incorrect and should be abandoned. In constructing a theory of the domain, we are not prescribing the allowed behaviour of the organisation - rather the assumption is that there is some (consistent, cohesive, complete and closed) set of rules that do govern its behaviour (that we are unaware of) and we are trying to discover and represent them explicitly. If a behaviour which the theory forbids is observed, then that theory clearly does not describe the rules that the system being observed obeys. Refutation of the theory thus takes the form of a search for behaviours that are forbidden by one of the 'rules' or theorems (be they axiomatic or derived) from which it is composed.

The way in which refutations are obtained is of necessity somewhat different when the system being investigated is a human organisation. Simple observation of phenomena is not generally sufficient to generate refutations for meaningful theories - if a theory is to have a chance of being correct it will have to account for the general case. That a theory can 'explain' an unusual and unexpected phenomenon makes it far more powerful, and it is consequently on the unusual, that the theory allows and yet previous theories and notions forbade, that we must focus our attention. In order to do this, conventional science relies on the use of controlled experiments. An experiment is an artificial environment that, assuming the theory is correct, constrains the system being investigated to behave in an unusual manner allowed by the theory. More specifically, if the theory is incorrect, a good experiment will force the system to exhibit 'forbidden' behaviours. Any theory will be limited in scope - a useful theory will not address aspects of the world that are of no immediate concern to us. A controlled experiment will exclude the influence of aspects of the world that are not covered by the theory. For example a basic theory of dynamics might not describe the behaviour of bodies in viscous liquids, so an experiment to test some aspect of it must somehow contrive to counteract the influence of viscous forces.

Where the domain being investigated is a human organisation, such controlled experiments become almost impossible. It is not feasible (nor morally acceptable) to inflict artificial conditions on the clinic just to test an analyst's theory. We might choose to rely on observation alone, but as with conventional science the most likely phenomena to be observed will be the most common - exceptions that prove the rule will happen less frequently, and may not be observed in the time frame of a reasonable development project. Even if we were willing to spend significant periods of time observing the organisation being studied, the resulting theory would only be of our own domain rather than that of the users or stakeholders of the system. The world view of the organisation developed by an external analyst will almost certainly be very different from that of a member of its staff.

Here we come to an important point: there is a fundamental difference between a theory of some part of the world conventionally talked about by 'hard science' and a theory of an organisation. When considering a physical system, such as an electric circuit or a particular molecule, each of us can be equally dispassionate about our contemplations. The same is not true of an organisation. The world view of those who are not associated in any way with the organisation might possibly be dispassionate in the same way as a 'scientific viewpoint' can be[xiii]: the world view of those who work for that organisation or are closely involved with it will certainly not be dispassionate. In a sense, the reality we are writing theories about is that belonging to a part of the system being analysed. It is for this reason that the concepts and categories that the analyst creates to understand the domain will probably be very different from those that the concerned participant uses.

In the domain analysis conducted in the Diabetes and Endocrine Directorate, direct observation was used relatively little - instead a form of 'experimental' interview was relied on. In these interviews, the analyst tried to elicit examples of behaviour of the domain from the interviewee - generally a clinician or paramedic. These interviews acted, after a fashion, as experiments in that the discussion was guided by the analyst to address areas of behaviour that would test the theory most rigorously. The theory is refuted when a counter-example to one of the theorems of the theory is produced by the interviewee. Through the use of interviews, guided by the predictions of the theory, we can test the theory in an efficient and effective manner. By eliciting anecdotal counter-examples from the clinicians being interviewed we gain two further benefits. Firstly we are building a theory of the clinician's domain rather than one constructed by the analyst: the clinician will relate examples using the concepts and terms that she is comfortable with. Secondly, by using a universal theory to guide the elicitation of anecdotal counter-examples, we are facilitating communication between clinicians and computer scientists.

When the scientific method was introduced, in Section 4.3, we saw that it involved two directives. The first was the insistence that a theory should be tested as rigorously and honestly as possible. This directive has been addressed through the advocated use of experimental interviews. The second was the need to render the theory ever bolder, or more falsifiable. The way this is done is by increasing the ratio of refutative potentially observable behaviours to corroborative potentially observable behaviours: in this way, a behaviour picked 'at random' from the set of all those that are possible would be more likely to refute the theory. We can either choose to restrict the allowable behaviours of models of the theory as it stands by introducing more rules over the existing structure, or we can introduce new components into the theory and then forbid some combinations of states of this new component and those of existing components.

These two methods of emboldening the theory can be illustrated with a simple example - the father-son relationship. Suppose we start with the theory that a person must be in the relation 'is son of' with exactly two people. We can restrict this theory by adding the rule that the identity of neither of the two distinct people that the first person is son of is the same as the identity of that first person. The first theory would allow a person to be his own son - the second, restricted theory forbids this. If the first theory had been tested to our satisfaction, we would attempt to refute the second by seeking cases where someone was their own father (though we would be surprised to find such a case). The theory can be enriched by adding a new concept: gender. We first expand the scope of the theory by introducing this concept, insisting that

[xiii] This is of course a simplification - we should take heed of Dunne's observation that 'No man is an island'. The interconnectedness of human organisations and systems is much less clear cut and the mutual influences much more insidious than for hard physical systems: there are thus few human systems that we do feel dispassionate about. Indeed the notion of dispassion about physical systems is moot (and objectivity which is often associated with it has certainly been challenged by the 'social construction of reality' stance taken here) but it would be fair to say that most people care more about their jobs or hobbies than the discovery of a new quasar or bacterium genus (although they may well be interested in this latter, they have less at stake than they do with the organisations they interact with).

every person has a gender, and that this gender can take one of two values: male or female. Although we have increased the number of possible behaviours, or rather states as this theory is presented as static, the ratio of allowable states to forbidden ones is preserved - we say that the new state component is currently orthogonal. The enrichment consists of introducing a new component and describing an interaction with those in the original theory. In this case we would say that the person who is the son must be male. By forbidding female sons we have reduced the number of states allowed, and thus the ratio of forbidden states to allowed states is increased *vis a vis* the original theory. Both these approaches were used extensively during the course of the analysis.

One of the capabilities of the Schuman-Pitt notation that was of great benefit in the progressive emboldening of the theory was that of theory composition. This was described in Section 6.2.9 - the schema calculus allows two separate class schemas to be combined to form a third which inherits the properties of both in a precisely defined manner. What this means is that sub-domains of the problem area can be analysed and described in isolation. The nature of the interaction of two sub-domains can be considered separately thus drastically reducing the complexity of the analysis. This technique was used extensively in the project, not least to consider the so-called 'specialisation' state components independently of the 'operational' ones, and then investigate the way in which the former constrained and controlled the latter. The investigation, representation and composition of these sub-domains is often called the 'separation of concerns' [Alex71] and good analysis methods support it.

The domain theory is the result of many iterative construction - emboldening - refutation - reconstruction steps. After eighteen months the process was stopped. It is not true to say that the resulting theory is perfect - what we can say is that it is better (bolder and more accurate) than it was at the beginning.

In the next section we will explore in a little more detail what an interaction theory is, and how it can be used to guide the engineering of the information system.

6.3.4 The Information System Specification

Once a satisfactory theory of the domain has been created, we use an interaction theory to tell us how well an implementation of a given information system specification will represent that domain. This begs the two questions - what is an interaction theory, and what is an information system specification. The latter theory is the more straightforward to explain so we shall consider this first.

The theory of the organisational domain represents a physical system by means of the rules that constrain its behaviours. If we choose a computer system as the physical system, and represent this through the use of behavioural rules, we will have an information system specification. Whereas the state components in domain theory should be interpreted as concepts in the real world (such as patients, doctors, clinical interventions and so on), the state components in the information system specification should be interpreted as aspects of a computer system. How these state components are to be implemented is an issue that need not concern us at this stage - we might subsequently decide that one state component should be represented as an entity held in a relational database, another as a global variable in the code of the system, another merely accessed remotely from another computer, and not stored in the local system at all. However the state components are implemented, they will exhibit, at an abstract level, the behaviours described in the specification.

The specification thus tells us how the state components of an information system are to interact - which combinations of behaviour and state of the different components are permitted. The specification tells us nothing about the domain that it is imagined that the information system will support. We can tell how

well a given information system implements the specification - we cannot deduce anything about the domain from these two entities so are ignorant as to the success or failure of the system in use. In order to do this, we need an interaction theory.

6.3.5 The Interaction Theory - What Is It?

The interaction theory tells us how the state components of the information system will be interpreted into the domain once the system is in use. A model of the interaction theory tells us what state or states of the domain a model of the information system specification represents. The elucidation of the interpretation of information system concepts is provided through the use of an interpretation function. A model of a given interpretation function takes the state of (a model of) a state component of the information system as its argument and returns the state of (a model of) a state component of domain. The interpretation function is not always a simple thing to represent as the state of the information system will affect its content in a non-straightforward manner. Some behaviours of the domain will not be represented by the information system, and some behaviours of the information system might be a simplification of what seems to happen in the domain. For this reason the interaction theory will not simply be a composition of the information system specification and the domain theory along with a number of functions: it requires the specification of rules over those functions as well. It is because there are forbidden states, or combinations of states, of the interpretation function that tell us that the two systems - the domain and the information system - interact. This is the reason this theory is called the interaction theory.

The interaction theory must be constructed as accurately as is feasible. As we saw earlier, it is almost impossible to use the scientific method to test this theory. We should thus devote much careful thought to the decisions made here - luckily for the most part these are fairly easy and uncontroversial. If we do manage to create a prudent interaction theory, we can use this to see how the state space of a model of the information system specification will be mapped via interpretation onto the state space of the domain. The approach we should adopt when constructing the interaction theory is discussed further in Section 11.3. Where this mapping is lacking or is in some way inadequate, there lies a potential weakness of the information system.

6.3.6 Engineering the Information System I: Arguments for Interpretational Weakness

The observation of a flaw in the interpretation of the information system into the domain does not automatically imply a problem with the specification as there are many reasons why we would not want to represent all the investigated aspects of the organisation described in the domain theory. Most obviously, there may be a stated desire (by the commissioner of the analysis) to support one aspect of the organisation but not others. Although this is generally the starting point for conventional starting point for systems analysis, here such a stated need is seen as a means of limiting the scope of the sub-domain which the information system supports. For example, having completed the work recorded in this thesis, the author suggested a number of areas where automation might be useful. Some of these components of an eventual information system were considered extremely useful (such as an expansion of the clinical record system, an integration with the hospital appointments system, and a contract management system), others were not thought to be immediately necessary (such as an internal referral system, or a system which could reconstruct medical records from computers in other departments). To complain that a system to manage the booking of patient appointments fails to account for the referring of a patient from one paramedic to another is to use a false yardstick - that part of the domain is not represented because it does not need to be supported (for now).

Not only will some aspects of the domain theory be unimportant to the sub-domain that we are currently interested in supporting, but even when the behaviour of the information system being considered is directly relevant to the problem being addressed, there are a number of causes which might lead us to justifiably depart from a truthful representation of the domain.

Firstly the more complex an information system, the greater the chance of error and the more difficult it is for the user to understand why that system is behaving as it is. For this reason simplicity should be striven for. Secondly it is a good idea to minimise the workload of the user. Many parameters and arguments are needed to accurately describe the state of a complex organisation such as the clinical directorate - asking the user to provide all of these might impose an unacceptable and unnecessary data entry burden. Thirdly, even within a well defined sub-domain where an information system is required, there will always be some areas where automated support is not required, and providing it will handicap the smooth running of the organisation in some way. For example, when a patient turns up with acute hypoglycaemia, she needs to be stabilised, not recorded. Lastly, we need to consider existing technical artefacts, whether they are the primitive data structures of the implementation vehicle, or legacy computer systems that are to be integrated: these all imply constraints on the behaviour of the system if it is to be affordably and efficiently implemented.

6.3.7 Engineering the Information System II: Arguments for Domain Conformance - The Developmental Motives

To set against the above quoted justification of faults in the interpretation of the information system are a number of factors that encourage conformity with the domain theory. These have been called the four developmental motives later on in the thesis. They are called motives as they motivate us to create an information system that is a more accurate representation of the domain. The developmental motives are summarised below (they are described in more detail in sections 12.4 and 13.4).

- gratuitous expansion of scope

If the scope of the domain represented by the information system is increased, we get a richer and more 'realistic' picture of the domain. If we can do this without significantly adding to the complexity of the system, the data entry burden on the user, or the difficulty of integrating the system with existing technology, then we should do.

- functionality of interpretation

For every state of the information system, there should be at most one valid state of the domain. In other words there should be no information system state that can represent (within its scope) more than one state of the organisation. If there were, the interpretation of the information system would be ambiguous and the quality of the information it stored would be degraded.

- reduction of behavioural entropy

For any state of a valid model of the interaction theory, the information system acting in isolation of the domain should not present to the user possible operations that are forbidden in the domain. We want to represent the constraints that are present in the domain and so help the user to avoid making mistakes in representation.

- prevention of prohibition

For any valid state of a model of the interaction theory, all the behaviours of the domain model in isolation must be allowed when it is embedded in interaction theory. The only way in which this might not happen is if the interaction theory defines areas of the domain where automated support is compulsory, and yet the information system is not capable of representing the totality of that sub-domain. An example of this somewhat baffling notion might be when an inadequate information system has been introduced and yet a parallel manual system is not allowed as it will lead to ambiguity.

6.3.8 Engineering the Information System III: How To Do It

Assuming we are interested in engineering a new system and not just examining an existing one, the above discussion still begs the question: how do we do it? Well, we are now in a position to put the pieces together. In the same way that the domain theory started with a hypothesised theory derived from educated guesswork, observations, and informal discussion with domain participants, so the system specification starts as a hypothesis derived from informed cogitations, observations and discussions with local computer specialists. For example, most 'new' information systems are integrations of existing legacy computer software and hardware - these must be specified separately and then composed together in such a way that the domain is roughly supported by the overall system.

Once the 'first version' specification has been completed, an interaction theory can be sagaciously constructed for it which reveals how it will be interpreted into the domain. Having done this, changes to the specification should be motivated on the one hand by the various reasons for divergence between the specification and the domain theory discussed in section 6.3.6, and on the other by the four developmental motives which encourage us to emulate the domain theory in the specification.

The process is iterative: having changed the system specification, a new interaction theory must be created, the affect of the changes assessed, and further modifications made to it as are appropriate.

Through the use of the interaction theory, the implications of the compromises made during this system engineering stage are revealed, and as a consequence the decisions that the analyst makes, in consultation with the would-be users and stakeholders of the system, become more enlightened.

6.4 Conclusion

This chapter described the method used both in terms of the notation with which the results were presented, and the processes and procedures used to derive those results.

The notation chosen was a state or model based formal notation. This decision was explained, and the way in which such an approach might describe an organisation - ie as an object with instantaneous state that is changed by 'events' - was discussed. The particular conventions and benefits of the Schuman-Pitt notation, the state-based formal method chosen were discussed briefly. The main benefit is the way in which theories can be easily built up out of simpler sub-theories so that a separation of concerns can be achieved.

The processes by which the final results - specifications of the appropriate information systems - are derived were presented. These hinge on the creation of three theories to describe the domain, the information system and the interpretation of the information system into the domain. Once these theories have been constructed (in the case of the domain theory through use of a variant of the scientific method) they can be used to engineer an information system with 'better' interpretational properties (if this is deemed appropriate).

We have discussed at some length how the problem area is to be investigated. What we have not yet done is consider what the problem area is. Although a detailed description of this is the role of the analysis process, and is thus presented in the third part of the thesis, a broad understanding is required before we can conduct any investigation at all. In short the boundaries of the problem need to be identified. These are discussed in the next chapter, in the light of existing analyses of the clinical area.

Chapter 7: Identification of the Problem Boundaries.

7.1 Introduction

All of the above discussions have revolved around the ability of a given method to analyse a particular problem area with a view to constructing a 'solution' in the form of an information system that provides support to that area. We have not thus far made any attempt to discern and name that problem area. To observe that the problem area is 'the clinical directorate' is not adequate - there are many aspects of such an organisation that we either cannot or do not wish to support with a computer system. Even those areas of the directorate's business for which we can and wish to provide automated support are extremely extensive: we could imagine clerical, managerial, financial, medical, personnel, resource management and many other types of computer system. To try and investigate all these areas of a clinical directorate in any depth is clearly infeasible in the time scale of the project. In short the scope of the initial domain analysis needed to be limited. Of course, any decision taken at the beginning of the project would be sure to be changed as the nature of the problem was illuminated by the light of experience. A preliminary identification of the boundaries of the problem area is nevertheless necessary if we are to apply our efforts efficiently. The statement of these boundaries is the subject of this short chapter.

7.2 Other People's Theories

Before we consider where the analysis ought to be focused, we should look at the results of analyses that others have done of similar areas - namely aspects of the provision of healthcare. In this section we will consider a number of such theories (although theories, these are generally referred to as models) and ponder the problem areas they are concerned with.

7.2.1 An Abstract Model of Care

Carson and Cramp [Carson93] present a highly abstract model which describes a generalised medical intervention using the control engineering metaphor of the feedback loop. This is given in the following figure:

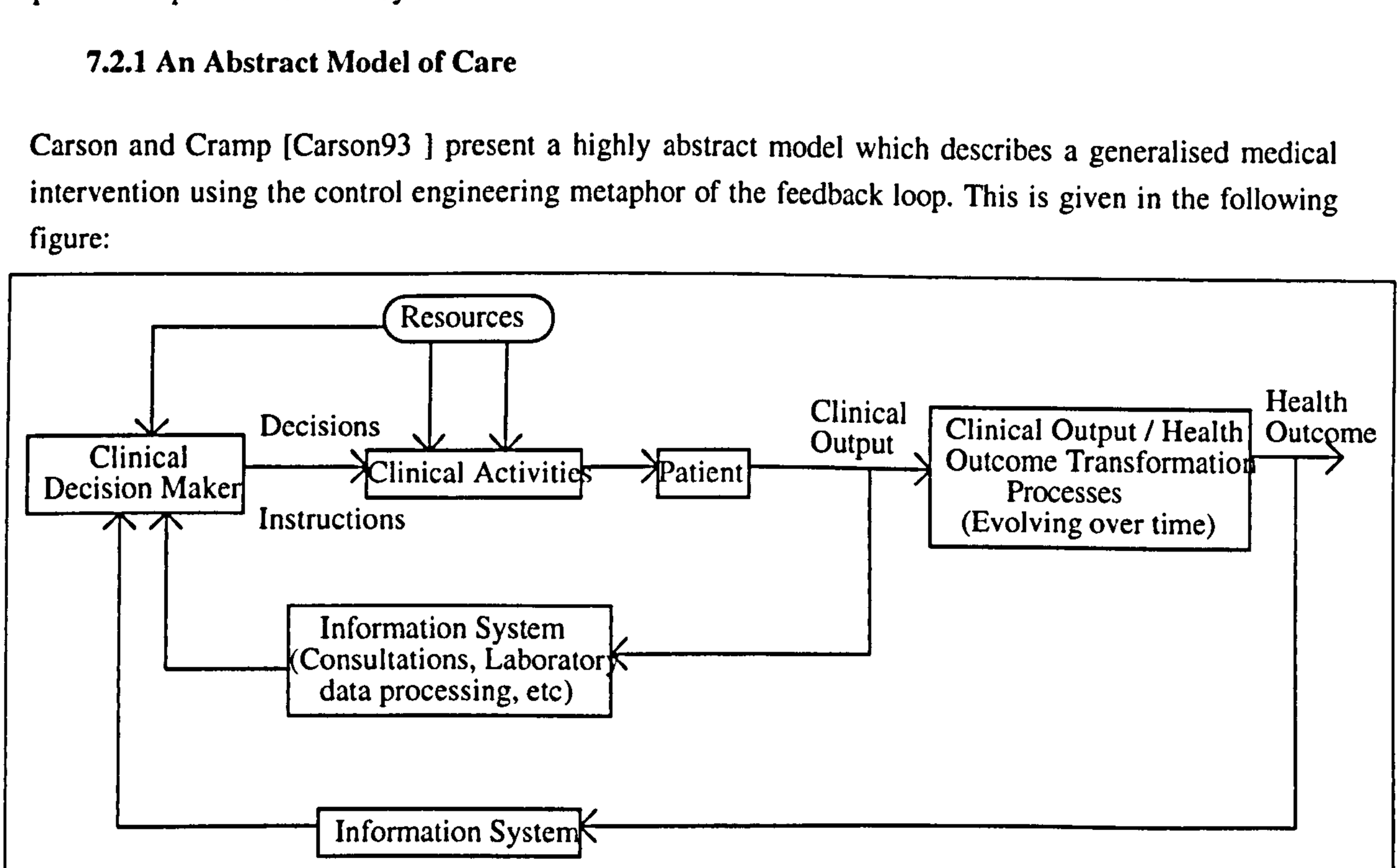


Figure 2-6: Feedback Loop representing the Clinical Process (From [Carson93])

In this model, the clinical process is shown as comprising a quartet of objects which act upon each other - namely the decision maker, the clinical activity or intervention, the patient, and information concerning the result of the intervention passed back to the decision maker by way of an information system. There are other objects which have a modifying effect on the process - these being the availability of resources

and the assessment of the merit of the outcome of the clinical process when compared against the expectations and desires of the patient, clinician, and society at large.

This abstract model is used to provide insight into the way in which a clinician intervenes to alter a patient's state of health. Its extremely abstract nature means that it tends to be used for didactic purposes to give a structured overview of the common elements of the medical process. The problem area in this case is the state of the patient's health, the medical decision making process, and the effect that the clinician has on the patient. Understanding of this kind is especially important if we are intending to create a system that will guide the clinician in choosing the intervention most relevant to the patient's condition. This is generally the role of an expert system rather than an information system. Because of this (and as we will see later the fiendish complexity of medical aspects of healthcare) the problem area described by this particular model is unlikely to be one that we are interested in.

7.2.2 A Customer-Supplier Model

Doyle [Doyle93] describes the procedures associated with the delivery of care in two settings, the Diabetic & Endocrine Day Centre featured in this thesis, and the Obstetrics ward at St Thomas' Hospital. One of these models is given below:

Supplier - Customer Relationships
eg. request for mid-stream urine (msu) test

1. Patient requests professional help
2. Doctor requests msu
3. Nurse takes urine sample
4. Nurse requests sample to be tested

5a. Clerk requests results
5b. Doctor requests bulletin board results
6. Doctor requests results in notes

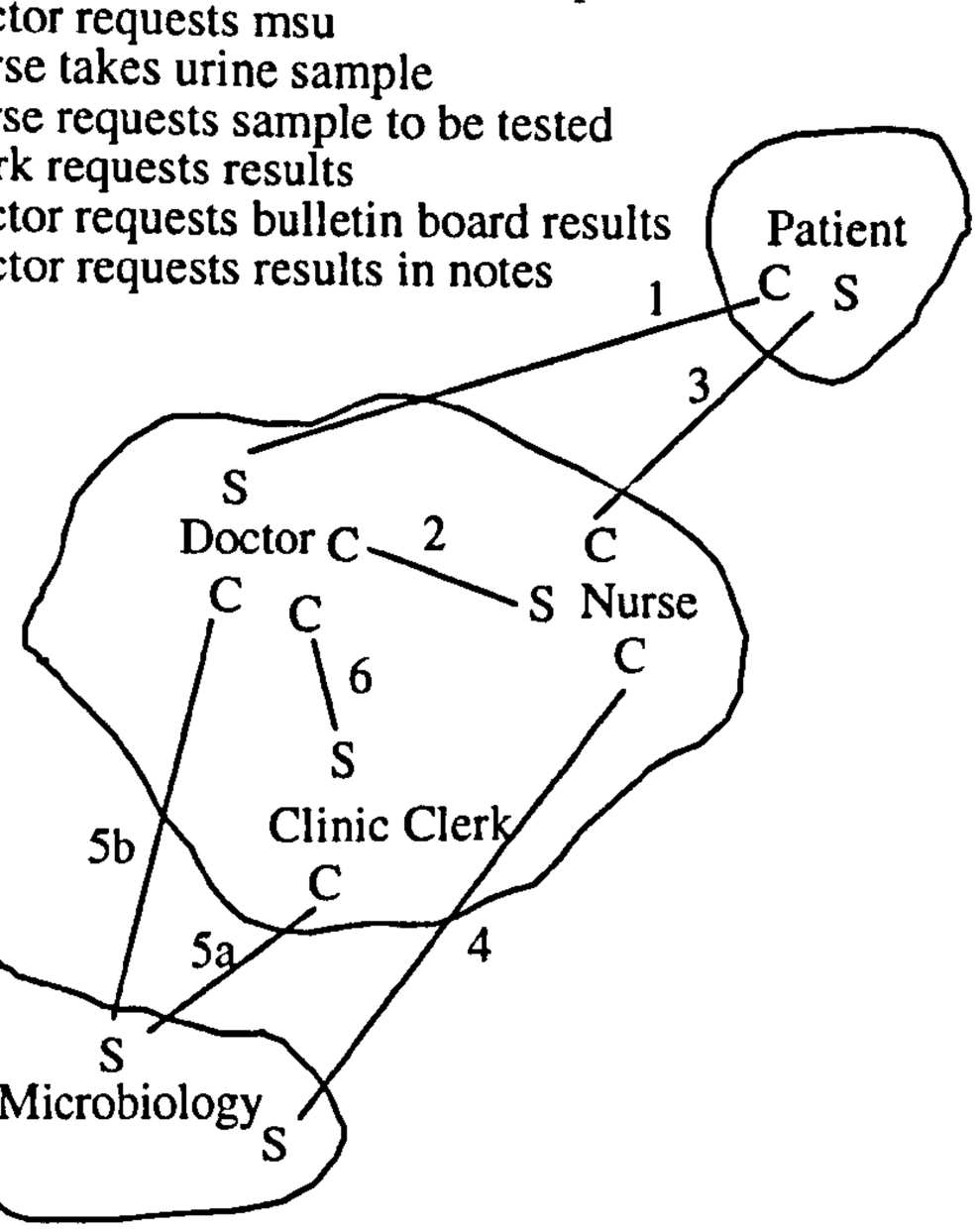


Figure 2-7: Customer - Supplier model of a procedure in the obstetrics ward. Taken from [Doyle93].

This diagram records the responsibility structure for the discharge of a particular medical process. It does this by representing the stakeholders in a particular clinical process, and explores the relations between them in terms of the service each provides, and the recipients of those services - in short who supplies what to which customers. For this reason these models or (theories) are called Customer / Supplier models. Models such as this are widely used by the Total Quality Management movement to help workers inspect and improve their jobs by thinking about the service they provide for others. We are not particularly interested in changing people's jobs, but we are interested in the jobs people do and how these

relate to each other. An information system supports an organisation by recording the changes wrought on it by the people that work in it and the external influences on it, so the nature and interactions of the tasks that domain members participate in is clearly of great interest to us. The boundaries of such a model might be similar to those of the system the project has been tasked with designing.

7.2.3 A Soft Model

Checkland uses work conducted by his department analysing some of the problems faced by the Community Medicine Department of East Berkshire Health Authority as an example in his book 'Soft Systems Methodology in Action' [Check90]. A simplified version of one such model (in the form of a 'rich picture') is presented below:

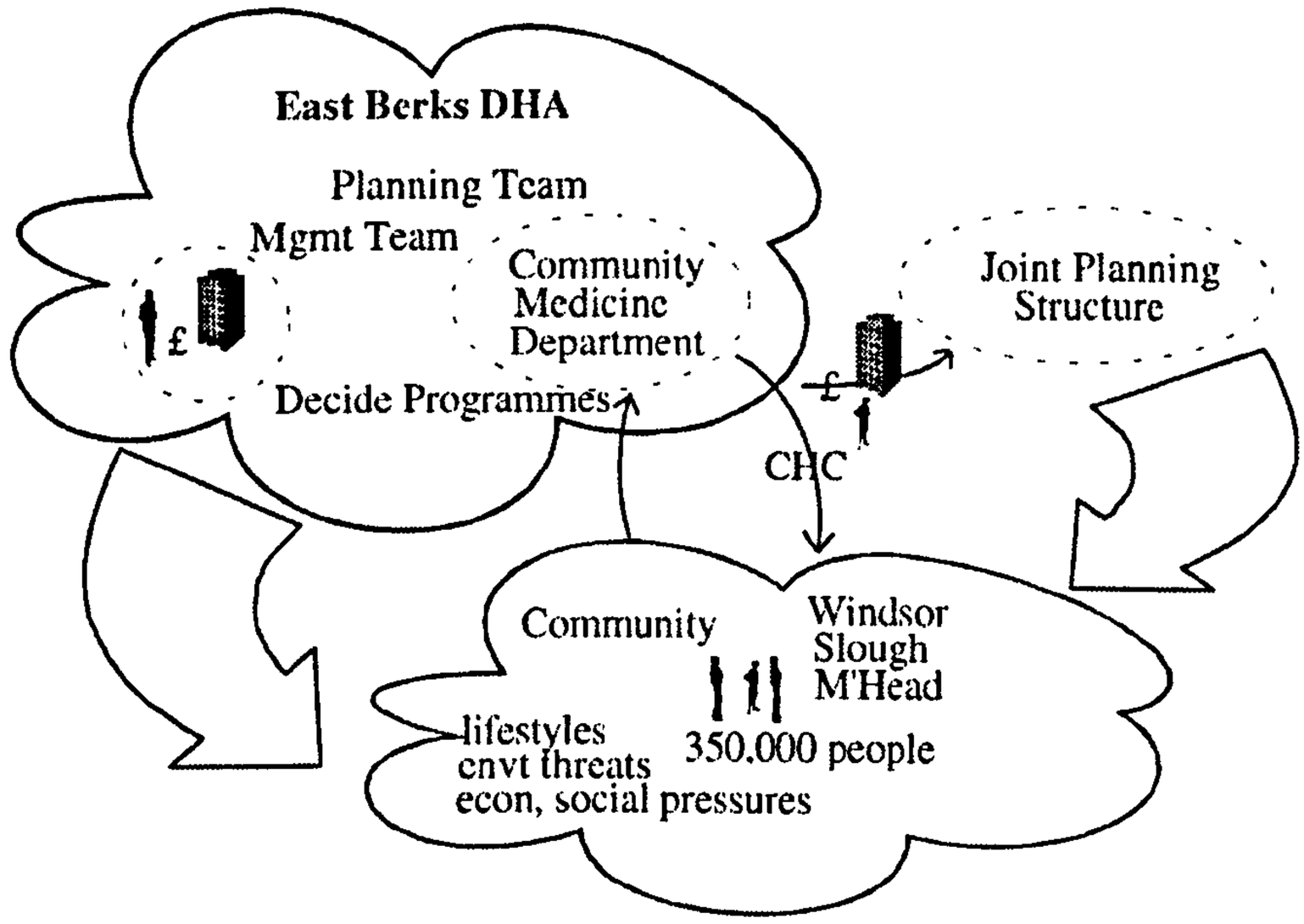


Figure 2-8: (part of) Rich picture for study of East Berkshire Health Authority

Models such as this are used to help guide organisational change: changing the procedures that organisations use and the relations between the individuals that go to make up that organisation. The boundaries of this model include inter and intra- organisational conflicts, managerial functions with respect to monitoring and control, and goals and purposes projected onto the organisation by its stakeholders. Although it must be recognised that the introduction of an information system might affect working practices in quite a major way, we are not interested in organisational change directly, and should not be surprised if the boundaries to the problem area addressed by this model are different from those that apply to the analysis for a Directorate Information System.

7.2.4 Clinical Data Models

In any hospital there probably exists a number of complex clinical systems used to store clinically relevant data in single departments or for individual doctors. By virtue of our assumption that information systems are interpreted into a problem domain when in use, we ought to be able to reconstruct some aspects of clinical domains from the theories embodied in clinical database systems. Reverse engineering a specification from an information system is extremely difficult: in the case of systems that use relational databases, we can extract an entity relationship model (or rather theory) quite readily. This was done for two clinical systems used at St Thomas' - ER diagrams for these are presented below.

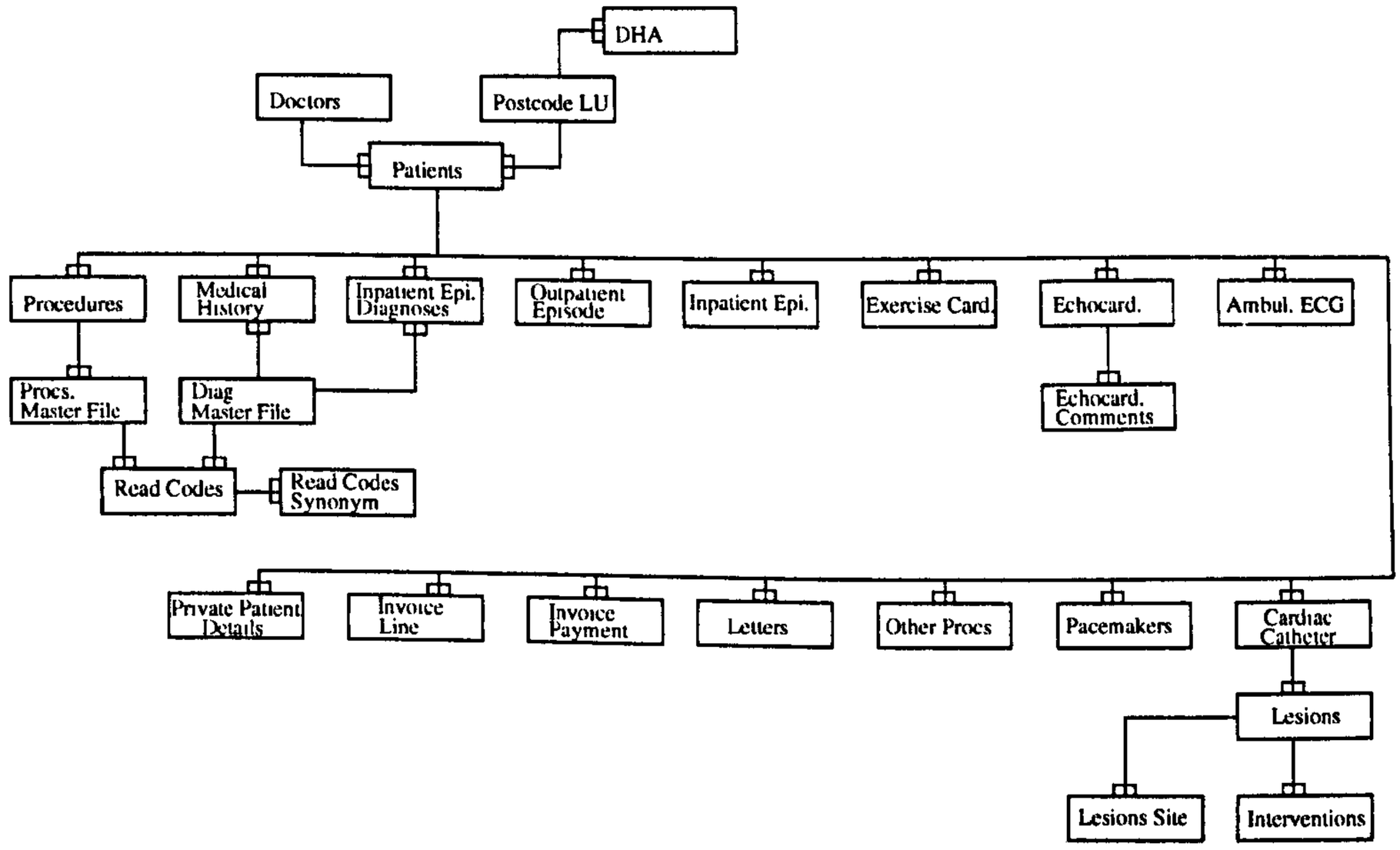


Figure 2-9: Cardiology System ER diagram

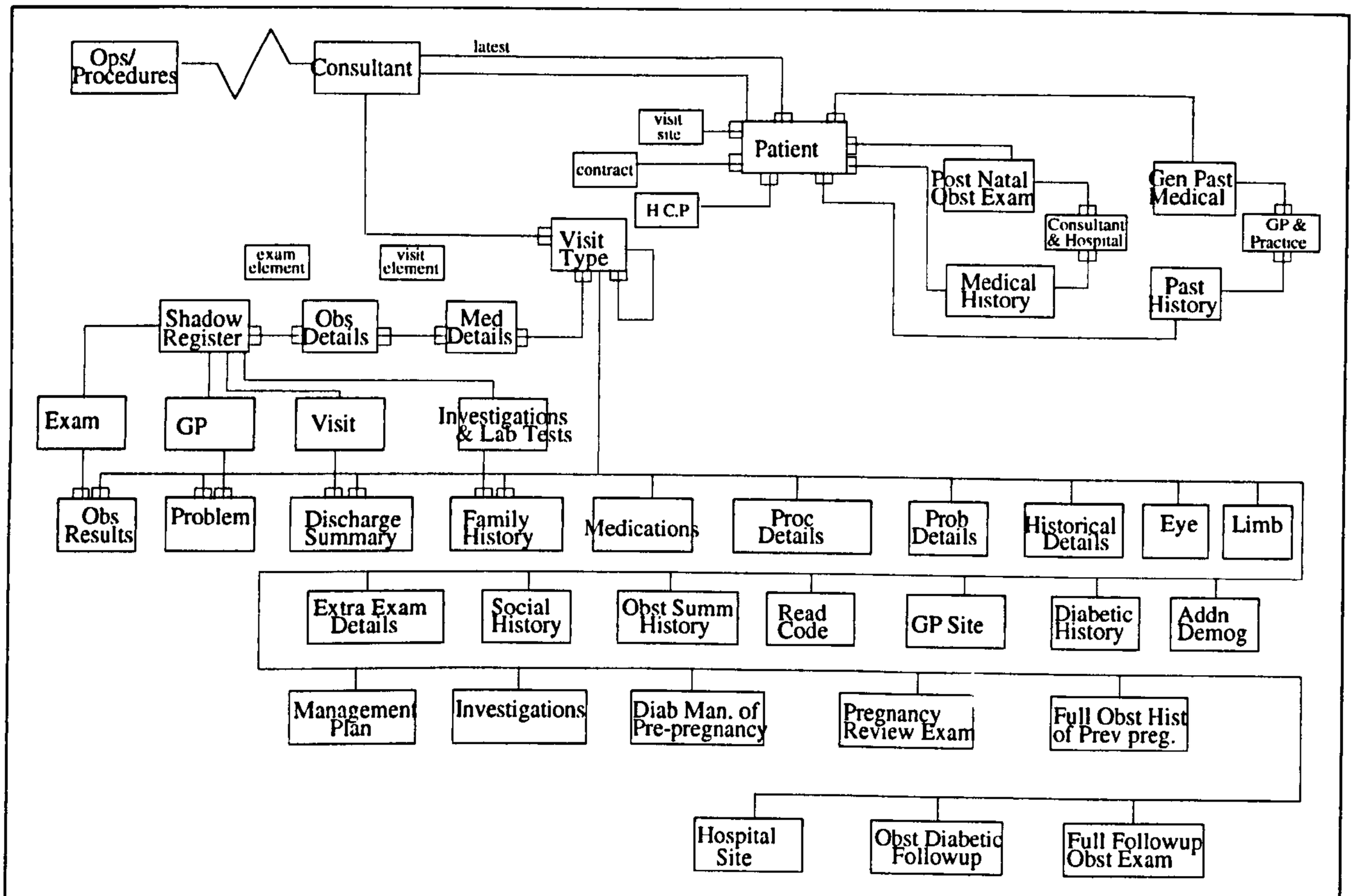


Figure 2-10: Early Version of Diabeta III System (Names of Entities have been changed to preserve product confidentiality)

Although there are some similarities between the two diagrams, the two models are very different and are both very complex. One of the goals of the experimental project set up by the hospital to analyse a Directorate Information System (of which this thesis is a report) was to discover and record the properties that such an information system would possess in all directorates - in other words the theory was to be generic not specific. The identification of an abstract data model of some value from the above two ER diagrams is a daunting prospect: before embarking on such a course, we should be careful that we do not bite off more than we can chew.

7.2.5 The Common Basic Specification

The desire to gain an understanding of the complexities of health care is not limited to academia and individual clinicians. The NHS itself has spent a great deal of resource and time over the last ten years developing a single data and process model that purports to describe the entirety of the health service of the United Kingdom. This process started in the mid 1980s with the Körner datasets and progressed to a more comprehensive treatment of the service in the form of the NHS dataset. Finally four million pounds and four years were allocated by the NHS Management Executive to a project to create a definitive model, called the 'Common Basic Specification' (CBS) [IMC92]. The history of the development of the CBS and its status as a theory of the NHS is presented in [Cohen93].

It was envisaged that all computer systems used by the NHS would be designed according to the rules recorded in the CBS. If this were the case, the scope of the model needed to be extremely broad: indeed the CBS is a truly vast data model, describing almost two hundred entities and sub-entities (for this reason, a pictorial representation of the model is not presented here). The 'specification' purports to cover all the activities that the NHS engages in, ranging from the most strategic policy decisions to everyday procedures carried out by doctors and nurses. The size of the CBS, and the effort that has gone into its

construction (hundreds of man years and millions of pounds) ought to encourage us to look for a less ambitious domain. If the envisaged use of the CBS were reasonable, then the fact that the Directorate Information System is to be a computer system used by the NHS, ought to enable us to find a description of the domain we are interested in somewhere in its extensive data structures. The reason why we should not use the CBS as it stands is that it has been designed in the same way as any data-model - non scientifically, with the results presented in a semantically poor notation. Because of this, we cannot be sure what a given concept means, and the absence of any refutative evidence means that we are ignorant of the justifications for the theorems embodied in the data structure. Still, as is discussed in Section 14.6, some of the concepts in the CBS seem to be very similar to those in the theory as developed.

The next section considers what an initial appropriate boundary might be for the problem domain pertinent to the construction of the Directorate Information System should be.

7.3 The Problem Boundary

7.3.1 Introduction

Before the project is commenced, there are certain assumptions that can be made which help to guide the analysis. These can be justified by considering the nature of the problem that the project is tasked with solving, and the issues raised by the review of other theories constructed in this area. One possible problem boundary is presented below and justified accordingly.

7.3.2 Operational Concerns, Not Managerial

Early on in the project it was noticed that there is a great cultural emphasis towards the 'grass roots' of the organisation. The basic operational activity is much more central to the essence of a hospital than is the case in many other organisations. Thus a hospital is commonly perceived as a physical and administrative framework within which medical care is provided to patients: a bank would generally not be perceived as merely a framework for the receipt and dispensing of money to individuals. This clarity of central function is reflected in the hospital's IT strategy document which states that 'management information should be a by-product of patient care' [KPMG89].

This argument is also supported when we consider the models described above. Where the problem domain includes patient management (as for example in the first model by Carson), the model is to be used for either for educational purposes or to guide the development of decision support systems. Where the problem domain includes resource and personnel management (as in the case of the model created by Checkland and the CBS), the purposes of the model encompass deep organisational change - by deep we mean that the controlling management structures are to be changed as well as the controlled operational systems. The purpose of the project reported here is the design of a computer system, not the achievement of organisational change although it is recognised that the introduction of information systems can have far reaching effects on the nature of the host organisation. As the purposes of the models whose boundaries incorporate patient and administrative management are different from the purposes of this project, we should not be surprised to see that these functions can and should be left out of the analysis.

7.3.3 Avoid Medical Details

The above argument tells us that the domain boundaries encompass aspects of the operational functions of the directorate, but exclude the managerial ones. This is still a phenomenally complex area embodying most of practical medicine. It was decided that the analysis would avoid considering the 'knowledge' side of medical care: at the same time as this project was being run, a patient record system was being

developed separately - by the DIABETA III project - that addresses many of the medical aspects of data storage for diabetes care. It should also be noted that much work is being carried out in this country and others in this area by the medical informatics community - for example by the AIM projects DILEMMA [Fox92] and GALEN [Rector93] - and it was important that the project did not duplicate any of this effort. Similarly the project avoided knowledge support as this has traditionally been the role of knowledge-based (or expert) systems, not information systems of the type envisaged.

Referring back to the clinical data models reviewed in Section 7.2, we are encouraged in our decision to avoid the temptation to represent medicine itself. The two data structures presented were fairly involved - the details of the data types not recorded in the ER diagrams mean that the composite properties of the database from which the diagrams have been extracted will be complex indeed. This intricacy is hardly surprising: medical information systems need a high degree of data resolution - the organisation of data needed to help with a particular clinical task tends to be extensive and highly interconnected: the information needs of medicine are very intricate and detailed. It is important to note that while all patient record systems are likely to be complex, they will be complex in different ways. Thus the Diabeta system and the Cardiology system display similar degrees of complexity, but their underlying data structures are very different: this is because they support very different tasks.

Because of the difficulty associated with the design and implementation of medical systems, we should be very wary of getting involved in this area. Similarly, the differences between the structures of the domains in different clinical specialties means that a generally valid theory of medical care would be even more difficult to devise. This is not to say that the search for the structure of a common medical record is a foolish one, only that we should not expect to be rewarded: the architecture of the general medical record is a form of 'holy grail' to medical informaticians. Indeed many large and sophisticated projects have endeavoured to define such a computerised record both in this country [Kalra93], and abroad (for example the emerging P1157/HL7 standard of the IEEE in the U.S.A.). Although contributing to the search, none has yet found what it was looking for.

As a result of the above arguments, it was decided that the perceived medical state of the patient, and diagnoses and decisions pertinent to that state, would be ignored by the analysis. In the event this restriction was complied with - even to the extent that the gender of the patient was not represented (this is only relevant to the medical process inasmuch as some conditions behave differently depending on the gender of the sufferer, and thus can be thought of as an unchanging state of health).

7.3.4 Be General and Accommodate Change

Finally, as was noted earlier, the project was set up to look at the problem of clinical directorates in general rather than a specific one. It was decided that the domain theory, although concentrated on the Diabetes and Endocrine Directorate should be able to describe (if possible) as many directorates as possible. The analysis needed to convey abstract principles rather than the particulars of individual departments. Yet at the same time the theory had to be able to say things that were useful to the department in question.

Even if it was decided to limit the project to the design of an information system for one directorate (presumably the Diabetes and Endocrine Directorate), abstraction and generalisation would still be of utmost importance. As we saw in Chapter 2: Historical Background to the NHS and St Thomas' Hospital, change is a constant companion to today's health service, and St Thomas' Hospital in particular continues to experience radical evolution and upheaval in the way it perceives itself and provides care. The failure of an information system to cope with change in an organisation is a common reason for system rejection

(indeed, often the organisation will change radically between the statement of the requirements and the delivery of the system thus rendering it obsolete before installation). The problems faced by the analyst who tries to ensure that the system specification will be able to cope with the changes in an evolving organisation are similar to those that he or she faces in endeavouring to render the specification equally valid in different organisations. In both cases the task is to find the common irreducible core behaviours that characterise the class of organisation. If this can be done, then a system based around that core should be equally useful in all organisations in that class - assuming that in its evolution a particular organisation stays in the same class (in other words we are assuming that the Diabetes and Endocrine Directorate, or a successor organisation stays in the business of providing health care), then the system should remain applicable.

However, it is important that the specifications devised are not only equally useful in a relative sense, but actually useful in an absolute sense. For this to be possible, the system must be able to support services that distinguish one directorate from another, or that characterise the evolution of a directorate over time: in short the analysis must acknowledge and deal with organisational idiosycracies rather than glossing over them through the use of abstraction. Yet how can a theory of an organisation be equally valid in all organisations (from a given class) and still describe the unique characteristics of any given organisation? We can resolve this apparent paradox through the identification of a common pattern to the idiosycracies of the different directorates, or the same directorate at different times. If we are successful in our search, then the different organisations (or the same one at different times) will display behaviours that are typical of one or other 'instances' of the pattern. The scientific method helps in our search for generality, and in testing any pattern proposed - it is down to the skill of the analyst to identify the pattern. This approach is taken in this project with the definition and use of two classes of state component: the specialisation state components which define the instance of the pattern, and the operational state component that record the organisation's instantaneous state. The pattern itself is specified in the form of the names of the state components and the invariant relations and mutual constraints between them (This concept is discussed further in Section 8.3.5). Although it might be moot to claim that the need for simultaneous abstraction and semantic richness is a boundary of the problem, it is certainly one of the characteristics of the problem domain and its possible solutions.

7.4 Conclusion

In this chapter we considered a number of existing theories that are relevant to the area of human activity that we are considering - namely healthcare. It was argued that the models that existed were either too abstract (Carson & Cramp's control engineering model), created for a different reason (such as to facilitate and guide organisational change as in the case of Checkland's model of the East Berkshire community medicine department), or too complex (the clinical data models of working clinical record systems). Armed with this review, we can consider what a reasonable boundary to the problem we are confronting might be. It must be clinically useful and thus specific to health care. It must thus focus on the work that clinicians do day-to-day rather than on the underlying nature of either the medical process (as in the case of Carson and Cramp's model) or the organisation of the delivery of that process (as in the case of Checkland's model). However, we do not want to get bogged down in the details of medicine as this is too complex (as evidenced by the data models for the clinical record systems described). Two guidelines consequently present themselves:

- we should focus on operational concerns, and not managerial ones; and
- we should endeavour to avoid medical details.

In addition to this, we have seen that the domain we are concerned with - clinical organisations - is changing very rapidly. This observation gave rise to a further guideline:

- the theory should attempt to illuminate what is common to all directorates, but still provide useful insight to individual cases.

The goals of the customer - supplier model and the CBS are similar to that of this project, and we should thus not be surprised to find common properties of these descriptions. For example, the customer - supplier model analyses processes in terms of types of clinicians and interactions between them (which in many ways parallel the HCP Type and Activity concepts), and the CBS has many concepts that have direct equivalents in the domain theory presented below. This is discussed further in Section 14.6.

In this part of the thesis, we first considered how all types of analysis could be characterised as using a limited set of processes and procedures, and presenting their results in standard forms. We then reviewed existing methods and the one chosen in terms of the processes used in the analysis, and the notation with which the results were recorded. The next part reports on those results and discusses the particular means by which they were derived case by case. Firstly we will consider the part of the hospital where the analysis was based, and give an informal overview of the way in which the theory attempted to describe the domain.

Part Three:

The Results

Introduction: Structure of Part 3

The objective of this part of the thesis is to report the conclusions of the analysis, both in terms of the theory of the domain that was developed and the subsequent investigations and designs of information systems using this theory. One of the main thrusts of the analytic technique is that of 'justification': it is important not only to present the results but also to explain how they have been derived, and why the particular structures settled on have been chosen. This section therefore attempts to give a justification both for decisions taken at the domain modelling stage and at the information system design phase. The formal theories of the domain, the various information systems and their interaction theories are not presented in their entirety here as this would obscure the messages that the author intends to convey. Instead only such fragments of the theory as are deemed necessary to help in the illumination of its most salient features are presented in the body of the text: the reader is referred to Appendices 2, 4, and 5 for the complete formal theories.

Part three is divided into eight chapters, which can be loosely aggregated into two logical groupings. Chapters 8, 9 and 10 together form the first natural grouping in that they all help to present the domain theory that was developed to describe the behaviour of a generic clinical environment. The formal arguments and justifications for the decisions taken are presented in chapters 9 and 10. The derivation of the formal domain theory represents the majority of the work recorded by this thesis. As such its exposition is careful, detailed and extensive (though the author hopes, not longwinded). In order to equip the reader with the necessary understanding before he or she can embark upon the discussion of the intricacies of the derivation of the formal theory, chapter 8 provides a brief informal overview of the ideas behind the finished article.

Chapters 11 and 12 explain different aspects of the relation between the domain theory and the understanding and / or the development of information systems that will act in that domain. Chapter 11 relates to an existing system - the Clinical Record System. The first part of this chapter describes the behaviour of the system, and explains how this was put into the Schuman-Pitt notation used in the project. The second part introduces a simple interaction theory and discusses how it can be used to explore the interpretational adequacy of the Clinical Record System.

Chapter 12 describes the development of a new information system, facilitated as it was through the use of an interaction theory. The first part of this chapter describes the structure and behaviour of the new system, and explains how it is (designed to be) composed of two pre-existing systems: the clinical record system (described in chapter 11) and an outpatient appointment system (described in this chapter). The system described, called DIS1 as it is a (first) fragment of a 'Directorate Information System', is thus an integrated information system. The second part of chapter 12 describes the way in which the use of the interaction theory guided the decisions taken in the design of the DIS1 specification. It is chapter 12 that explains why and how the domain theory is useful in the design of computer systems and is thus crucial to the understanding of the contribution of this thesis to the discipline.

The structure of part three is summarised in figure 3-1 below.

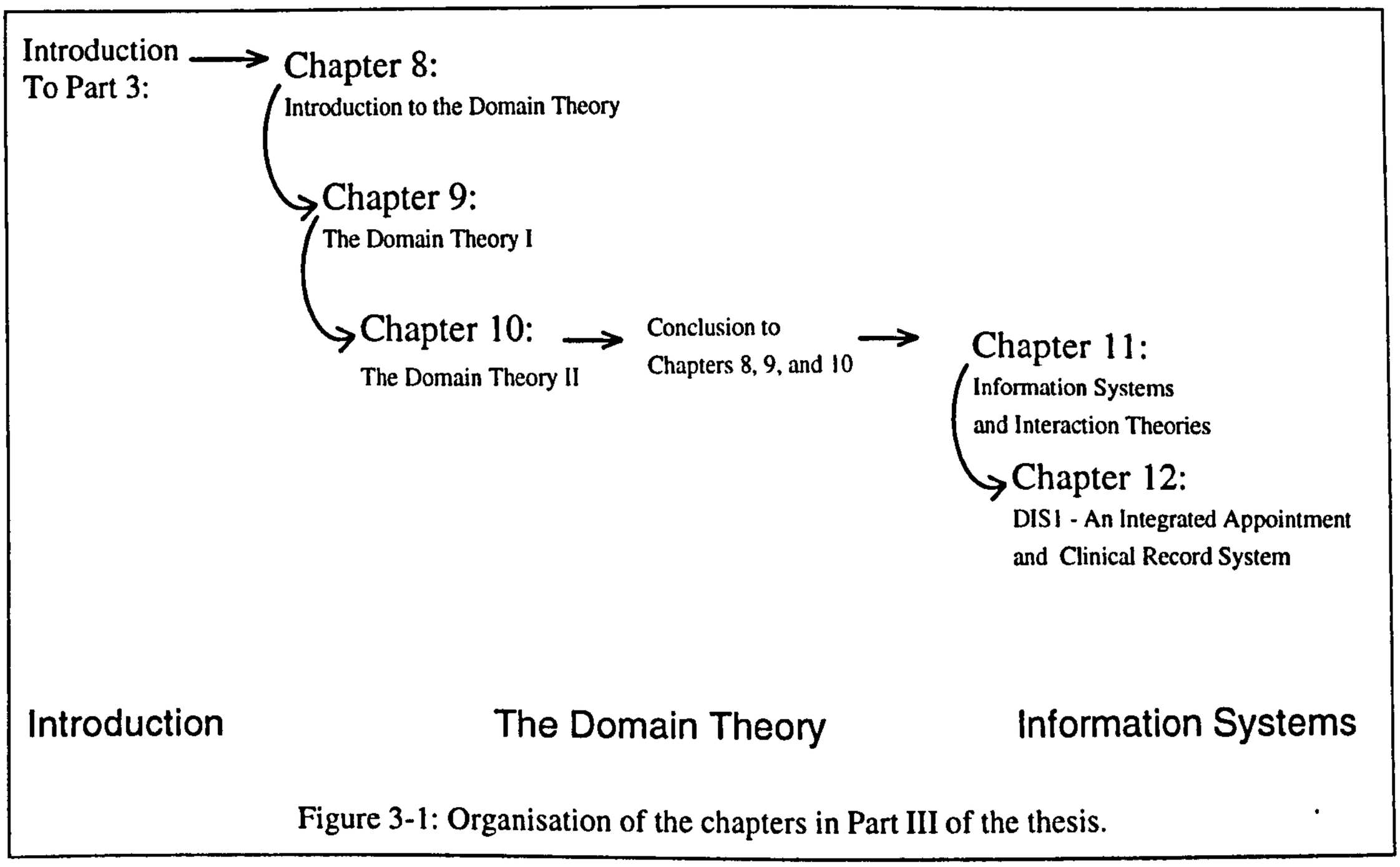


Figure 3-1: Organisation of the chapters in Part III of the thesis.

The presentation of the theory relies on the use of italics to record the names of formal quantities. Italic terms with upper case initial letters are the names of state components - sets, relations, and other structures - those with lower case initial letters are generic names for arbitrary members of models of those state components. Thus the term *Activities* denotes the basic set whose members in the domain are 'medically meaningful encounters', while the term *activity* represents an arbitrary member of that set. Occasionally, when talking about various theorems of the domain, we need to distinguish between different members of a model of a state component, or incorporate a name for one or more such members into a set-theoretic expression. In this case we might say something like 'consider an *activity* $a1$' whence $a1$ is a specific, named member of the set of which *activity* is the arbitrary, unnamed member: that is to say *Activities*.

Chapter 8: Introduction to the Domain Theory

8.1 Introduction

8.1.1 Presentation of the Domain Theory

The purpose of the next three chapters is to explore the theory of the medical domain that was developed by the author over the course of the project. This phase of the work lasted eighteen months and the concepts presented are thus the majority of the 'results' of the project. By the end of chapter 10, the reader should not only be furnished with a detailed understanding of the domain theory as it currently stands, but also have an understanding of why the theory is as it is, and how the assertion that it represents the medical domain might be justified. The domain theory itself, expressed in the Schuman-Pitt notation employed in its development, is presented in Appendix 2.

To enable this, the reader is first given a brief overview of the theory. This is the purpose of chapter 8. A basic grasp of the form and scope of the theory is required before any more detailed exposition if the reader is to understand what is presented and not get lost in the detail of the subsequent presentation. Once we have seen loosely how the theory works, we can attack it in more detail.

Chapters 9 and 10 examine the domain theory in more detail. As we have already discussed, the development of the theory was influenced heavily by the 'scientific method'. The presentation reflects this, being organised around two directives of the method: the quest for a refutable and bold theory, and the reconstruction of the theory on refutation. The first directive results in the creation of ever more complex and semantically rich theories of the domain. This is represented in the following description through the incremental introduction of state components (in the form of sets, relations, and other set-theoretic structures), operational refinements and class compositions, all of which imply movement from simple and sparse to complex and rich forms of representation. This enrichment of the theory is done in such a way as to make refutation more likely: in the absence of refutation, the bolder theory is the more useful (this was discussed earlier in Section 4.3). The notation used supports this incremental enrichment of structure through the composition of smaller, or more primitive, conceptual units or 'classes' to form more complex aggregate classes, models of which are in some way behaviourally constrained.

While this search for semantic richness and behavioural constraint is the strategy guiding the exposition of the theory, the tactics are those of refutation. Consequently the majority of the contents of the next two chapters thus follow a similar pattern. A property of the theory is introduced and described both informally and formally: a refutative counter-example observed in the domain (or rather elicited from a clinician who observed the counter example in the domain) is described and shown to disprove the theory: and a new property introduced to replace the old that is not refuted by the counter-example.

Many of the lessons learned through the development of the theory reflect not only on the structure of the theory but on the nature and appropriateness of the scientific method that was employed. These are briefly discussed as and when they arise in the description of the theory, and rounded up and discussed more fully in Chapter 13.

8.1.2 Introduction to This Chapter

The theory as developed is large and complex, and spans over 40 pages of set theoretic notation, but the central concepts can be quickly explained. The most important abstract entities are activities, types and patients which are represented in the theory as sets, and the (instantaneous) interaction between these

entities as relations between the sets. These sets and relations are all 'state components' whose values together describe the state of the domain as it is at any stage. Remember that it is the state of a model of the theory rather than the theory itself that provides this description (see the earlier discussion on the distinction between a theory and a possible model of that theory given in Section 6.2.3).

Although the theory was created to be capable of representing a large number of directorates, the project was based in one in particular: the Diabetes and Endocrine Directorate. This behaviour of this directorate influenced the nature of the theory more than any other, and it would be fair to say that insofar as the theory is broadly applicable, it is a generalisation of a theory of this 'home' directorate. Although the concepts from the theory are introduced below in as explanatory a manner as possible, the reader will find them easier to understand if he or she has a basic knowledge of the Diabetes and Endocrine Directorate. To this end, the section below describes some of the services delivered by this part of the hospital.

8.2 The Diabetes and Endocrine Directorate

8.2.1 The Endocrine Disorders and Diabetes

The Diabetes and Endocrine Directorate exists to provide care for patients with endocrinological disorders. These are many and varied, but are all metabolic in nature, and all are concerned with problems relating to the endocrine glands, responsible for secreting certain hormones into the bloodstream. Common types of endocrine disorders include acromegaly, Cushing's syndrome, goitre, a number of growth and development disorders, and diabetes (more correctly termed diabetes mellitus to distinguish it from other forms of diabetes).

Diabetes is the most common and complex of these disorders, and is thus dealt with in a more sophisticated way than the others. In general, however, the endocrine part of the directorate works in much the same way as a normal medical department - patients are seen by the specialist doctor for a number of out-patient visits, prescribed medication or other form of therapy, and if necessary admitted to hospital for further tests and possible surgery.

Diabetes care is dealt with differently, partly because the disease is more complex, and partly because it is more common than other forms of endocrine disorder. Diabetes mellitus is one of the commonest and most widespread Chronic Disease, and it affects between 1 & 3% of the population of the countries in Europe. It is a life-long condition that can be controlled but not cured. When their condition is well controlled, people with diabetes are healthy and fully active in virtually all walks of life. When poorly controlled, it can and does cause substantial morbidity and early death. Particular problems associated with the condition are blindness (Diabetes is the commonest cause of blindness in those of working age), limb amputation (Diabetes accounts for more than 25,000 limb amputations each year in Europe), and renal failure. The key to diabetes care lies in regular screening for problems so that they can be checked and solved before serious complications evolve. This involves close co-operation between many different types of health care professional, at all levels of care, but if the care is successful, the patient need never be admitted as an inpatient.

8.2.2 The DEDC: Collaborative Out-Patient Care for People with Diabetes

The following section describes the nature of outpatient care as it is delivered to diabetic patients by the DEDC. It is based closely on some previous work conducted by Shirley Smith, the physiologist at the directorate, which was summarised in a document intended to market the out-patient part of the

directorate, the Diabetes and Endocrine Day Centre (DEDC) [Smith92]. As such, the document is a succinct introduction to the services provided.

People with diabetes fall into two groups. The most common form is treatable through the prescription of appropriate diet and tablets, and affects older patients: this is called non-insulin dependent diabetes mellitus (NIDDM). The less common, but more well known, form strikes people at a younger age, and requires insulin injections as well as diet and tablets for treatment. This is known as insulin dependent diabetes mellitus (IDDM). Care differs for the two groups.

There are a number of health care professionals who work in the DEDC. These include:

- Doctors with specialist training in diabetes and endocrinology
- Diabetes specialist nurses with special training in the education and treatment of people with diabetes
- Diabetes dietitians with expertise in diets which are fundamental to the treatment of diabetes
- A chiropodist with expertise in foot care for diabetic people (foot disease is a major complication of diabetes)
- A physiologist with expertise in the testing for eye disease and nerve disease, two more areas where serious diabetic complications might be manifest
- Clinic nurses who perform a range of tests
- A laboratory technician who conducts the chemical analyses relating to some of those tests

A formal diabetic clinic is held on most working days in the week. The service that such a clinic provides for the patient depends on whether they have IDDM or NIDDM, and whether they are new patients (to the day centre) or followup patients.

New patients with IDDM are often emergencies in which case they will be seen by a doctor briefly immediately, who will instruct the specialist nurses to start the patient on insulin. An hour long appointment with the doctor will be made for all new IDDM patients, though for emergencies this first visit is arranged at the earliest opportunity possible. The specialist nurses will then see the patient several times in the first week, and with decreasing frequency for the next few months. New IDDMs are seen as soon as possible after diagnosis by the dietitian for one initial and two followup consultations.

Once they are registered with the clinic IDDM patients are seen for followup appointments with the doctor once every 6 - 8 months: alternate visits taking the form of the more thorough 'annual review'. If any problems are detected, the doctor may refer the patient to one of the other specialists in the centre. At present, followup visits are continued indefinitely, or until the patient dies or moves away from the area.

New patients with NIDDM are seen initially by the doctor for a detailed examination (sometimes lasting over an hour). The patient will then typically be referred to the specialist nurses in order to attend group patient education sessions. The patient will attend two such sessions which are attended by a specialist nurse, a dietitians and the chiropodist. Patients also see the dietitian for an individual appointment, reflecting the crucial role diet has to play in treating this condition.

NIDDM patients are then seen for followup appointments with the same frequency as IDDMs, and similarly are referred to other specialists as and when the need arises. These specialists include a chiropodist and a physiologist.

Patients with an acute foot problem, perhaps an ulcer, that can nevertheless be treated as outpatients will be seen by the chiropodist once a week for several weeks until the problem is resolved. Other patients that do not have an immediate problem, but are considered to be at risk of foot complications by the doctor are seen once a month by the chiropodist. All patients are checked annually for developing eye complications. This is done by the physiologist in the Diabetes Eye Complication Screening clinic. This involves the photographing of the retina of each eye, and examining the resulting print for certain patterns which indicate incipient eye disease. If discovered, the patient will be referred to the hospital's central eye clinic for laser treatment which should solve the problem before it causes more serious damage (such as blindness).

There are a number of more specialised clinics run by doctors in the day centre. These include: the children's clinic, run jointly with the paediatric directorate; the MARS clinic to treat diabetic related impotence; and the combined antenatal and diabetic clinic run jointly with the Obstetrics and Gynaecology directorate.

8.2.3 In-Patient Care

All the services described above are conducted by the DEDC as part of the care given to diabetics who are outpatients of the hospital: sometimes diabetics are admitted as inpatients. This will happen in two cases. Either the patient is admitted as an emergency via the Accident and Emergency department, or for elective treatment which requires residence in the hospital.

People with diabetes are occasionally admitted as emergency patients if they have had some form of extreme reaction caused by the condition. Such patients will generally be exhibiting the symptoms of hypoglycaemia (insufficient glucose in the bloodstream), hyperglycaemia (over abundance of glucose in the bloodstream), or ketoacidosis (caused by excessive ketones and acetones in the bloodstream). In advanced cases, such patients might arrive in a coma. These people will be admitted as inpatients where their conditions will be brought under control as quickly as possible. After this they will be seen as outpatients, and if not already one of the clinic's patients, registered with the day centre.

If the strategy of avoiding complications through regular examination and screening fails, then the patient might need to be admitted electively to the hospital. This might be in order to amputate a limb that has developed gangrene, or provide kidney treatment if the patient is suffering from renal failure. Assuming the patient recovers, they will be discharged and continue to visit the DEDC as an outpatient. If the patient is suffering from one of the other endocrine disorders then she might be admitted for other types of surgical treatment (such as the removal of part of the pituitary gland) or for some of the more involved clinical investigations.

8.2.4 Other Directorates

Although the majority of the analysis was conducted within the Diabetes and Endocrine Directorate, several other areas of the hospital were considered briefly. These included the following units responsible for providing aspects of medical care.

The General Myeloma Clinic: This is a standard outpatient clinic run for people who suffer from cancer of the bone marrow. At this clinic the doctor decides which investigations to request, and what actions should be taken on the basis of the results of those investigations.

The Haematology Contact Clinic: This is a clinic run by the haematology department for patients with well understood and routine problems. For example the patient might be recovering from a thrombosis and need to be prescribed the blood thinning drug wharfarin - either the drug will be prescribed at this clinic or the GP will be instructed to continue the writing the prescriptions. This process can be conducted by post with the patient represented by a 'log book' that they send in.

The Dermatology Directorate: This is one of the most complex directorates in the hospital, having been a separate hospital until relatively recently. All types of skin disorder are treated here, there being sixteen types of specialist clinic (with different clinics covering skin tumours, urticaria, dermatitis, blisters, hair, nails, corns and so on). A patient referred to this directorate will initially attend the general dermatology clinic where their condition will be assessed: if necessary, they will be referred to the appropriate specialty skin clinic. Each specialty clinic will provide treatment as is appropriate, or will call on one of the services available centrally to the entire directorate (such as use of the sun lamp, and provision of dressings).

The Diabetes Directorate at the Medway Hospital: One of the research doctors at St Thomas' is the Diabetes consultant at the Medway Hospital in Kent. It was thus possible to investigate briefly how another Diabetes directorate differed from the Diabetes and Endocrine Directorate at St Thomas'. The care provided by the department at Medway is in many ways more sophisticated and represents a model of what the service at St Thomas' might become. In particular, much importance is placed on the development of shared care, where responsibility for the patient's condition is shared equally between the GP and the hospital clinic (at least while the patient is an out-patient). Typically the GP will hold a 'mini-clinic' for all her diabetic patients once a week or month. The GP might be assisted by a specialist nurse from the hospital, and possibly by the consultant as well. The patient can also be referred to the hospital if they seem to be developing complications that the GP does not feel qualified to deal with, and once there can access a multitude of services in much the same way as with the DEDC at St Thomas'. This shared care system is seen by many as the way in which diabetes care will develop over the next few years: the high quality of specialist care will still be available to the patient, yet expensive hospital visits will become less frequent thus saving money.

Having considered the nature of the directorate where the analysis was in the large conducted, we are in a position to examine the theory that was developed. As was discussed earlier, before considering the formal theory, an informal introduction is presented to help in the reading of the next two chapters.

8.3 The Domain Theory: An Informal Overview

8.3.1 Introduction

In this section I have attempted to explain the meanings of the state components to help in their interpretation by the reader. The way the sets interact, and how they behave and constrain each other is the essence of what the theory describes. Exactly what is meant by a particular set can never be fully 'pinned down' or described from one person to another without any fear of ambiguity, but a mixture of informal description such as that given below and the formality of the theory together support the process of comprehending exactly what is meant by, for example, the set Patients. The problem of interpretation is profound and is discussed at some length in the conclusion, in section 13.3.

One of the central themes of the theory is the interaction between 'operational' state components, and 'specialisation' state components. The main form that this takes is the constraints that the values of the set *Types* and its graphs place on the possible behaviours and values of the set *Activities* and its graphs. This idea and its attendant problems are discussed at some length in the next few sections, but the reader should be aware before the theory is described of the importance and centrality of the interaction to the representation of medical practice that has been produced, for it is here in this distinction that the theory is able to represent the particulars of a given directorate and yet remain general to all.

8.3.2 *Activities*

An activity is a medically meaningful encounter between a patient and some representative of medical care. This encompasses obvious encounters between a clinician and a patient as well as more abstract medical activities such as the care delivered to a patient by the clinic over a period of years. Examples of activities are:

- The meeting between Dr Lowy and Mr Jones at 4.30pm on 26th May 1992
- The meeting between Sister Kidd and Mr Jones at 4.50 on 26th May 1992
- The meeting between Mr Jones and Dr Smith six months later.
- The care provided to Mr Jones by Sister Kidd over a prolonged period starting in May 1992.

The first three activities are straightforward meetings between a healthcare professional and the patient but the third is an example of a more abstract activity - the continued provision of care by one healthcare professional to the patient. *Activities* can stand in a number of relations to each other. One activity can be *Before* another, and one activity can *Include* another. Both *Before* and *Includes* are represented in the theory as partial relations. Thus an activity may be *Before* a number of other activities, and may include a number of others.

The relations *Before* and *Includes* are intended to be medically as well as chronologically meaningful. Thus a blood test for Mr Smith might well be required *Before* a followup visit for the same patient, in which case the blood test will have to be completed before the followup visit starts. A blood test for Mr Smith might turn out to be scheduled to take place before a followup visit for Mrs Jones: although these two activities will be chronologically ordered, this is not the intended meaning of the relation *Before* as expressed in the theory. Similarly with the *Includes* relation - if activity $a1$ *Includes* activity $a2$, the intended interpretation of this situation is for $a2$ to be a part of $a1$, rather than just for $a2$ to start after $a1$ starts and to finish before $a1$ finishes which would be a more strictly chronological understanding.

In the theory as it has developed, the concepts *Before* and *After* are not much used. They are left in as they are used in the development of the idea of the followup activity and might be developed further in future. Although it seems that medically related chronological ordering is not particularly useful in the representation of the Diabetes and Endocrine Day Centre, it might very well be important in other areas of medical care, particularly for the modelling of 'Care Profiles' and 'Care Protocols' which are a mixture of guidelines and 'best practice' rules for nurses and doctors (generally respectively) to use. The further development of these ideas is discussed in the conclusion of the thesis.

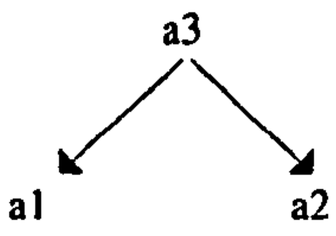
An activity can be requested, whence it becomes a *Request*, can then start, whence it becomes a *Proceed*ing activity, and can then finish, whence it becomes a *Complete*d activity. *Request*, *Proceed* and

Complete are disjoint subsets of *Activities* - the operations described move an activity into *Request*, from *Request* to *Proceed* and from *Proceed* to *Complete* respectively. *Request* and *Complete* are more 'abstract' concepts than *Proceed* (which is itself a fairly abstract notion) but they constrain the state of the domain as perceived just as much as do those activities that are currently running (though not in the same way). The way we should think about requests is as potential activities - for example an appointment to see a patient at some time in the future, or merely an agreement to have a patient referred for treatment. By treating Requests, Proceeding activities and Completed activities in the same way, we can examine their life cycles: an activity changes state rather than nature as time passes.

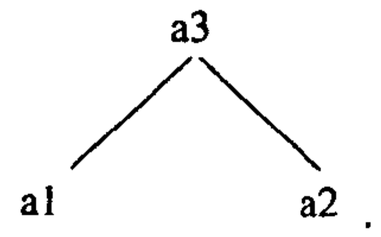
8.3.3 An Aside: "Graphical Graphs", a Helpful Notation Introduced

In general, instances of graphs can be represented just as precisely and much more clearly using a graphical notation. This is introduced here: consider the partial relation *Includes*. This is a graph as it is a relation over a single set, so in the case of *Includes* both the domain of the relation and the range, or codomain of the relation both come from the same set, that is the set of activities.

Suppose we were investigating a possible model of the theory, and were interested in the activities $a1$, $a2$ and $a3$ where $a1$ represents the meeting between Sister Kidd and Mr Jones at 4.50 on 26th May 1992, $a2$ the meeting between Sister Kidd and Mr Jones at 11.30 on the 17th of October 1992, and $a3$ the care provided to Mr Jones by Sister Kidd over a prolonged period starting in May 1992. We might want to say that $a3$ *Includes* $a1$ and $a3$ *Includes* $a2$ (In fact, we probably would want to say this). The model might record this as a textual representation of the relation as a set of pairs with the set defined by listing each of its members. Thus the model of *During* would be $\{(a3, a1), (a3, a2)\}$. An alternative graphical model of the same relation might represent the pairs as arrows between elements in the basic set (ie the set of *Activities*). A graphical model of *During* would then be



or more simply still



In the second graphical model of the relation, the arrow heads are dispensed with and replaced by the convention that all lines are taken to be arrows pointing down the page. In both cases we can see that the left hand arrow represents the pair $(a3, a1)$, and the right hand arrow the pair $(a3, a2)$. I will use this notation sporadically from now on where it might clarify an otherwise difficult point.

8.3.4 *Types*

A type is the medical description of the activity: in other words, what sort of an activity is it? Some types are:

- Doctor Consultation
- Specialist Nurse Consultation
- Dietitian Consultation
- Ophthalmologist Consultation
- Specialist Nurse Care.

We can assign each of the earlier mentioned activities to a type. Thus the meeting between Dr Lowy and Mr Jones at 4.30pm on 26th May 1992 is an activity of type Doctor Consultation; the meeting between Sister Kidd and Mr Jones at 4.50 on 26th May 1992 is an activity of type Specialist Nurse Consultation; the care provided to Mr Jones by Sister Kidd over a prolonged period starting in May 1992 is an activity of type Specialist Nurse Care. Each activity has exactly one type, a fact we express by defining the relation that returns the type of an activity, called *ActType*, to be a total function.

When an activity is created it must be given a type which it keeps from then on: an activity cannot change type.

As explained earlier, we have tried to carefully disentangle the specific properties of one area of hospital care (at one period of time) away from the general properties of all areas (or one area as it changes over time). The theory is general and abstract in structure, but can be specialised to represent a particular clinic or part of the hospital. This is done by giving values to the types described above, and to relations between those types. For example the specialisation to the diabetic day centre involves types such as 'Diabetic Care', 'Diabetic Specialist Nurse Consultation', 'Diabetic Eye Complication Screening Consultation' and 'Oral Glucose Tolerance Test', whereas a specialisation to the Dermatology department involves types such as 'General Dermatology Clinic', 'PUVA Visit', 'Dressing Session', and 'Urticaria Consultation'. This issue is discussed in more detail in the next section: Structure and Value: Different Levels of Refutation.

The distinction between activities and types, and the static and dynamic relations between them are central to the domain theory. Rules and structures over the set of types dictate how activities that are of those types can behave, and what possible activity structures are or are not permissible.

One example of such an invariant linking types to activities is that concerning the relation *Can_include* (this relation has been replaced by a more sophisticated one called *TypeGuide* which will be discussed in depth in the body of the chapter: the ideas underlying both relations are the same, and the simpler version is sufficient to explain the concepts involved for now). The relation *Can_include* is a graph over types so each member of the relation is a pair of elements from the set types. In the specialisation for the Diabetes and Endocrine Day Centre the pairs (Diabetic Care, Doctor Care), (Diabetic Care, Diabetic Specialist Nurse Care), and (Diabetic Specialist Nurse Care, Diabetic Specialist Nurse Consultation) are all members of the relation *Can_include*. The invariant linking this relation to possible activity structures says that if an activity *a*1 *Includes* activity *a*2, then the type of *a*1 *Can_include* the type of *a*2, or to put it another way if the pair (*a*1, *a*2) can only be an element of the relation *Includes* if the pair (Type of *a*1, Type of *a*2) is in the relation *Can_include*.

Using the graphical notation based on that introduced in the previous section, we can represent part of the model of the *Can_include* relation that records the nature of the type structure in the Diabetes and Endocrine Day Centre, and see how this affects possible models of the *Includes* relation.

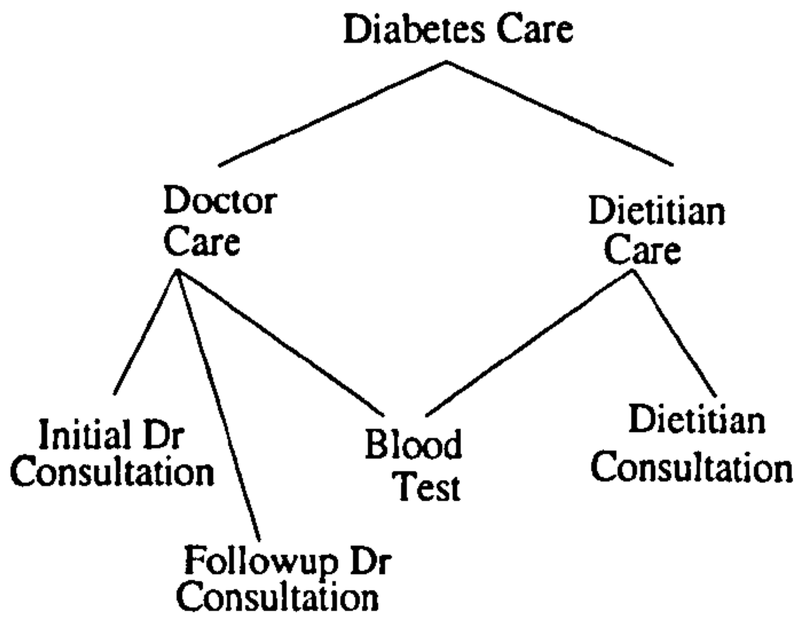


Figure 3-2: Model of the *Can_include* relation.

This model represents the part of type structure in the Diabetes and Endocrine Day Centre. Here each node represents an element of the set of *Types*. Note that both Doctor Care and Dietitian Care *Can_include* Blood Test

Using this as our model for *Can_include*, we can see that the following graph of *Includes* is permissible:

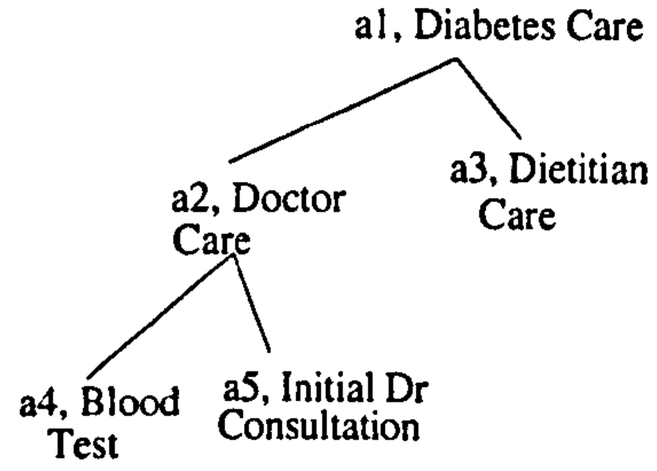


Figure 3-3: Permitted Model of the *Includes* graph.

This model is one permitted by the model of *Can_include* described earlier. Here each node represents an activity, as identified by the label to the left of the comma. In addition, the type of each activity has been recorded to the left of the comma - we can do this as each activity is associated with one and only one type.

(in fact this is a model of the graph:

$(id[Activities] \lozenge ActType) \circ During \circ (id[Activities] \lozenge ActType)^{-1}$)

We can see that this model is permitted by the rule relating *Can_include* to *Includes*, assuming we use the model of *Can_include* described above. For example, activity *a*2 *Includes* activity *a*5: this is permissible as *a*2 is of type Doctor Care, and *a*3 of type Initial Doctor Consultation, and Doctor Care *Can_include* Initial Doctor Consultation. On the other hand, the following model of *Includes* is not permissible

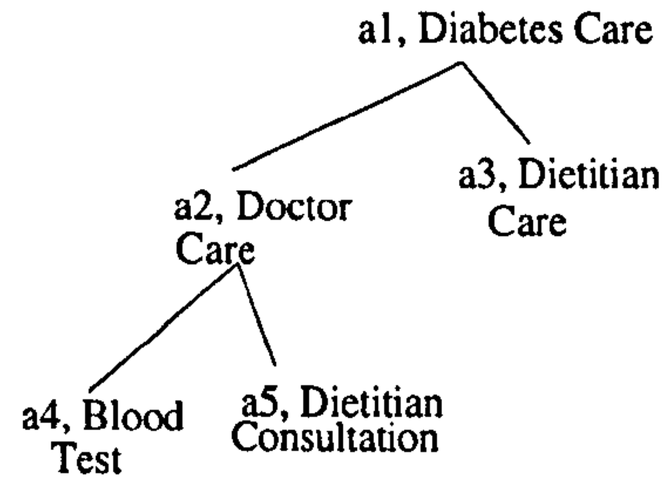


Figure 3-4: Forbidden Model of the *Includes* graph.

This is a model of the *Includes* graph forbidden by model of *Can_include* described earlier.

We can see that this model is not permitted by looking at the activity pair (*a*2, *a*5). Activity *a*2 *Includes* activity *a*5: *a*2 is of type Doctor Care, and *a*5 is of type Dietitian Consultation, but the pair (Doctor Care, Dietitian Consultation) is not in the relation *Can_include* so this model is prevented by the same rule as allowed the previous model.

There are a number of other rules relating possible activity structures, and behaviours of those structures to types: that linking *Can_include* to *Includes* is perhaps the most important.

8.3.5 Structure and Value: Different Levels of Refutation.

As we have seen behaviour of activity structure, the operational record of what the clinic 'does', is governed by both rules linking *Includes* to *Can_include* and the particular model of *Can_include* we are using. The rule is intended to apply to all (or many) areas of medical care, and is specialised through the instantiation of the set of types and the relation *Can_include* to reflect the behaviours and structures observed in a particular area. We might think of the instantiation of the set of types and relations over them as a configuration of the general theory so that it fits a particular medical area.

We can think of the theory as being expressed at two levels. The first and most abstract theory is one that applies to all areas of medicine: a 'theory of care' perhaps. The second is a more specialised theory that applies to one clinical area, for example the Diabetes and Endocrine Day Centre, which is different from the general theory in that it has been partially instantiated - the 'configuration' sets and relations have been given values. The more specialised theory is still a theory rather than a model as most of the sets are still to be assigned values. Interviews with doctors and other clinicians only allowed the specialised theory to be refuted: in some cases this meant that the general theory had to be incorrect (if the refutation covered a 'theoretical' property such as activities and relations between activities) while in other cases it might be that the general theory had been configured poorly and that the models of the specialisation sets were 'incorrect'. The general problem in the latter case was that there was no efficient way of deciding that the error was one of instantiation or of general theory structure, so although a new, and possibly more 'correct' theory could be constructed there was no guidance obtainable from the refutation to decide which of the two theories - the general or the specialised - should be reconstructed.

The way the analytic technique we used coped with these problems was to first try and re-specialise (or configure) the concrete theory while keeping the general theory the same, and then if no specialisation

was capable of rendering models that behaved in the desired manner, the general theory was re-worked. Examples of this approach are given in the section which describes the theory in more detail.

It should be noted however that the problem was especially acute as by far the majority of effort was spent in only one clinical area: the Diabetes and Endocrine Day Centre at St Thomas' Hospital. Although other clinics and hospitals were looked at, there was not enough time to examine sufficient departments in sufficient depth to enable the different levels of error mentioned above to be easily distinguished.

This difference between general and specialised theories crops up throughout the analysis. I will call the sets and relations that specialise the theory so that it represents a particular medical area specialisation state components, and those that record the instantaneous state of the clinic or department operational state components. Using this parlance we can say that *Activities* and *ActType* are operational state components whereas *Types* and *Can_include* are specialisation state components. The difference is one of stability: although we can expect the types of activity delivered by a department, and the types (and names) of clinicians employed to change over time, this change will take place over a much longer period than the change in the state of activities currently Proceeding. New activities are started every few minutes, new clinicians are employed every few months.

The theory as it currently stands does not explore how the configuration state components behave dynamically: the introduction of a new type of activity would be represented by a re-specialisation of the theory. The theory does describe static rules that must hold true of the configuration state components - for example a type cannot be in the relation *Can_include* with itself (in fact we say that the graph *Can_include* is directed and acyclic) - but does not describe how the state components will change. This is not to say that this is issue is unimportant: on the contrary the ability to support change is one of the most fundamental requirements of a medical information system. It is merely noted here that the problem is very difficult and only partially addressed by the theory. This is one of the areas where more work might be fruitful, as is discussed in Section 14.7

8.3.6 An Example of *Types* and its Graphs: The Diabetes and Endocrine Day Centre

As has been stated previously, the majority of the work was based in the Diabetes and Endocrine Day Centre at St Thomas' Hospital which is where all the outpatient activity of the Endocrinology directorate takes place. In order that the reader can get a feel for the way the theory dealt with this environment, a model of some of the specialisation state components is given here as it was applied for the day centre. The earlier and simpler version of the type structure is given here - the more sophisticated representation is given in the detailed description of the domain theory.

The majority of the clinical activity in the directorate concerns diabetes which is where the analysis was initially focused. The most abstract type of activity as it applies to the service the directorate provides for diabetics is called 'Diabetes Care' in this specialisation. Diabetes Care can be discharged in a number of ways by a number of people. In the day centre a number of different types of professional act in concert to provide that care: these are Diabetologists (which we call Doctors in this specialisation), Diabetic Specialist Nurses (DSNs), Dietitians, Chiropodists, Ophthalmologists and Clinic Nurses. All these types of clinician, except for the Clinic Nurses, are responsible for delivering care in certain well defined semi-autonomous areas of professional expertise. For this reason the types Doctor Care, DSN Care, Dietitian Care, Ophthalmologist Care and Chiropodist Care are all represented in this specialisation, and activities of these types are all candidates for inclusion in activities of type Diabetic Care. Each of these types of care are discharged through successive consultations between the health care professional and the patient concerned. Thus we say that Doctor Care *Can_include* Doctor Consultation. Included also in an activity

of type Doctor Care might be Blood Tests and Telephone Calls. We can see a similar situation with the other paramedics (DSNs, Dietitians, Chiropodists and Ophthalmologists). In the case of Doctor Care, it was felt that the initial consultation between the patient and the doctor was sufficiently different in kind from subsequent, or followup, Doctor Consultations. We can express the types and the graph *Can_include* as follows:

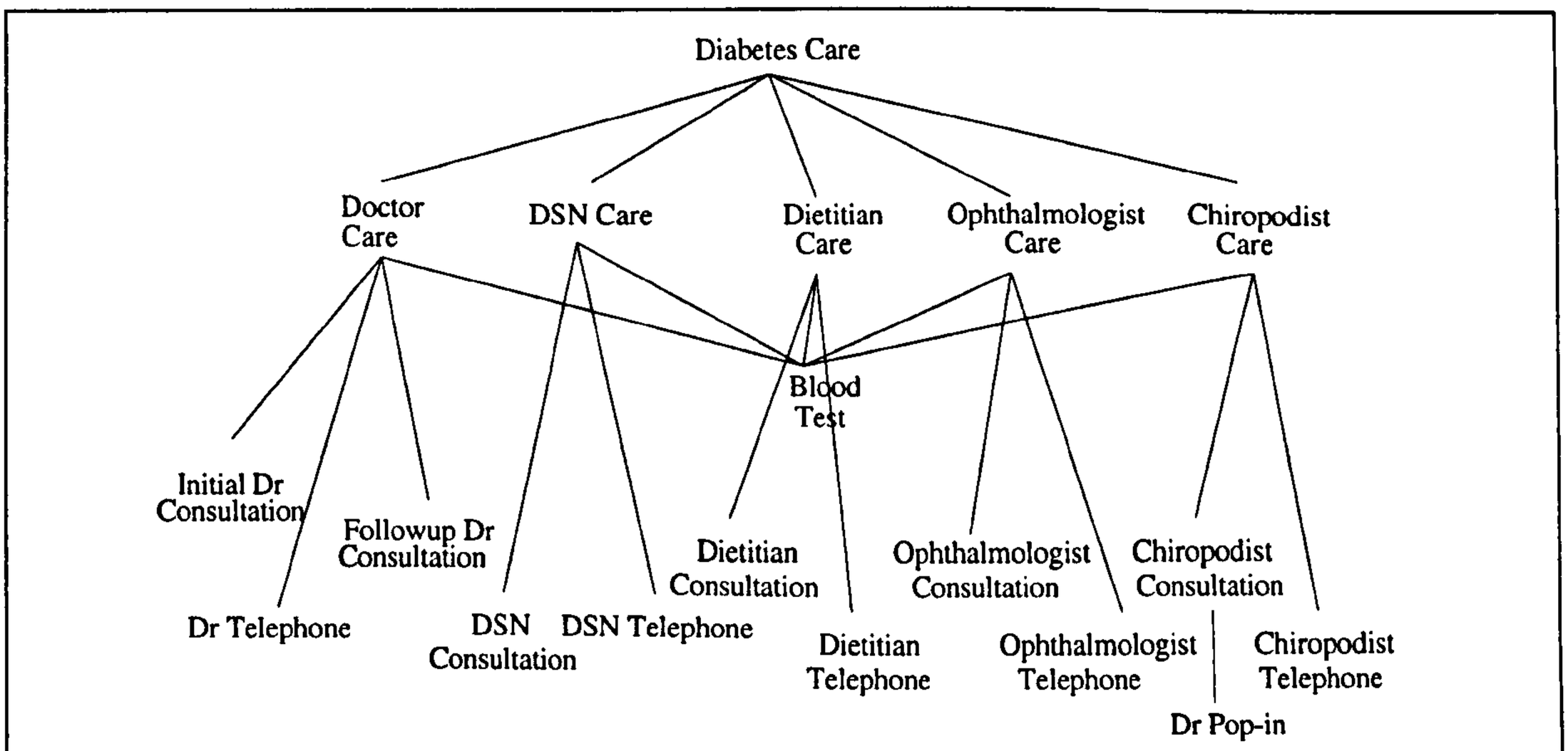


Figure 3-5: Model of the (directed acyclic) graph *Can_include* as specialised for the Diabetes and Endocrine Day Centre.

Occasionally the chiropodist will want to prescribe a drug to the patient. As, in the day centre, the chiropodist is not qualified to do this, she must call on the doctor to briefly agree that the prescription is appropriate and sign the prescription card. This type of activity has been called the 'Dr Pop-in' to emphasise its brevity: it is in the relation *Can_include* with Chiropodist Consultation.

8.3.7 Patients

A patient is any person who is a 'target' of a medical activity (see Section 13.3 for a discussion of how concepts can reinforce each others semantics on interpretation). All activities must pertain to exactly one patient. We express this fact by defining the relation that returns the patient for whom an activity is created, called *ActSubject*, to be a total function. Although every activity has one patient as its subject, that patient need not be present for all or any of the activity. For example a patient need not be present for her blood to be analysed, though she must be present for the blood to be taken in the first place. Similarly a patient need not be present for an activity of type Specialist Nurse Care to run, though she must for the component specialist nurse consultation activities.

If two activities are in the *During* or *Before* relation to each other then they must have the same patient as their subject. This is a corollary to our earlier insistence that these two relations must be medically meaningful. The development of this constraint is described later in section 9.3.3.

The set Patients and function *ActSubject* are operational state components.

8.3.8 Clinicians

A clinician is the medical person responsible for the delivery of care. Thus a clinician can run an activity, or can request for someone else to run the activity. Clinicians are grouped by clinician type, examples of

which in the Diabetes and Endocrine Day Centre are Doctor, Dietitian, Diabetic Specialist Nurse, Chiropodist and so on: each clinician has only one type, so this relation is represented by a total function, called *ProfType*. A type of activity can only be run (ie an activity of that type started such that it becomes a Proceeding activity) by clinicians of a specified type. This is recorded in the relation *RunType*. Similarly different types of activity can only be requested by given clinician types. This is given in the relation *EmbedType*.

In the same way as the set of types, and rules over those, clinicians and types of clinicians are specialisation state components: although the theory contains rules for deciding whether or not a particular structure of clinicians, clinician types, activity types and so on is possible it does not explain how these change over time.

8.3.9 Time & Information

Concepts representing time are introduced towards the end of the theory development. Time is modelled as a sequence of labels. For example one label might be 6.11 pm, 6 October 1993, and the 'next' label in the sequence might be 6.12 pm, 6 October 1993. All activities are given a time when they were requested, when they started and when they finished. In addition appointment times are given for some activities.

Towards the end of the theory's presentation, medical information is introduced and described. Of course, the whole theory is of 'information' in a sense in that its models are a representation of the domain. Information as a set in the theory is more specific than this however, and refers to information recorded about the patient, during a particular activity such that the record is in some way long-lasting. This covers not just the 'official' medical record, but also the various notes that are jotted down and passed between clinicians, test results, the records that professionals other than doctors keep about their own patients and so on. Information is associated with activities through the relations RecSource and RecCont in such a way that a record is linked to a single piece of information, and every record relates to only one activity.

The theory uses an utterly unstructured representation of medical information: The content and format of the medical record has not been investigated at all. This reflects the earlier decision to leave the 'knowledge' side of medicine well alone. Many other workers have looked at, or are looking at, the structure of a generic medical record and no firm consensus has as yet emerged. Medical record systems specific to particular departments have been created and are used to great effect but these tend to be very non-generic. By representing information as a simple set, we can allow for any subsequent structuring of the medical record, and we thus preserve the generality of the theory. The simplicity of representation acts as a sort of boundary to the problem area: the decision not to describe the health of the patient is reflected by this aspect of the state of the organisation being represented as abstractly as possible. Thus no insight is given as to the structure of a patient's health, or rather the organisation's perception of the patient's health, other than to observe that it is appropriate to consider each patient separately, and that certain operations change the value of that perception, albeit in unspecified ways.

8.3.10 Realism

The basic concepts introduced so far are suitable for the description of a (useful) part of the characteristics of a particular medical area, and its instantaneous state. As we shall see, the operations that are used to represent the behaviour of the organisation are flexible and describe valid changes in that instantaneous state. They can do this because they are general, primitive and can be composed together to form new operations of greater complexity: they do not on the whole represent operations that might be recognised in the domain. For example, the operation *.Embed* would probably not be one that clinicians clearly

understood: they would probably talk instead about referrals, followup visits, appointment booking and so on. Towards the end of the theory, we compose, refine and categorise the primitive operations so that their names can be more realistic, and they can describe more directly the 'business activities' of the directorate.

8.4 Conclusion

This last chapter presented aspects of the directorate where the project was based, and introduced the domain theory in an informal way. The theory that was created was intended to be generic, but most of the analysis was conducted in the Diabetes and Endocrine Directorate. It might thus be fair to say that the theory is an abstraction of one derived from an observation of one particular directorate. However, a number of different areas of St Thomas' and a department in another hospital (the Medway hospital in Kent) were also investigated in order that this abstraction had some credibility.

A brief description of the services offered by the directorate is given, both in terms of out-patient care and the rarer in-patient care. The patients in the directorate can be divided into those that suffer from Diabetes Mellitus and those that are afflicted by a different endocrinological disorder. Endocrine patients are treated in outpatient clinics and sometimes admitted to hospital if surgery or complex test are required. Diabetes patients are given sophisticated care mainly by the collaborative care team in the Diabetes and Endocrine Day Centre as out-patients. They will be admitted to hospital as emergency in-patients if their diabetes becomes dangerously out of control, or electively if they need kidney treatment or require the amputation of a limb.

The subsequent section in this chapter introduces the domain theory so as to aid comprehension of the formal chapters which follow. The theory is complex, but the main points are quickly explained - these centering on the interaction between *Activities* and *Types*. A distinction is made between specialisation and operational state components. The specialisation state components have a constant value for any particular model of the theory. These describe the structure and behavioural rules as they apply to the particular medical organisation being modelled. The operational state components record the instantaneous state of the particular organisation and thus characterise observable behaviours. *Activities* and its graphs are the main operational state components: types and its graphs the main specialisation state components. The distinction is introduced above, but is covered more fully in the next two chapters: while reading the subsequent chapters it is important to be clear as to the difference between the general theory, a specialisation of the theory, and an operational model of a specialised of the theory.

Other fundamental concepts described include clinicians, patients, time, information (or rather an abstraction of the state of the patient's health, or more accurately still of the organisation's perception of the patient's state of health). This chapter also introduced a pictorial representation that can be used to talk about graphs. This form of representation will be used and expanded on in the next two chapters.

Chapter 9: The Domain Theory I

9.1 Introduction

This and the next chapter give a more detailed description of the domain theory and provide some justification by way of refutations of early theorems and their replacement with unrefuted theorems.

This chapter describes those concepts that are the most fundamental and essential to the working of the theory. The next will describe enrichments to these fundamentals that make the theory more 'realistic' through the introduction of new state components (such as Patients and Health Care Professionals) and more refined operations (such as Referral and Booking).

The chapter is divided into four parts. The first introduces the central concept of the theory represented as the set *Activities*, explains the various partitions of the set and how their members behave dynamically. The second part describes relations between activities which represent medically meaningful ordering and composition of activities. The third part introduces the first specialisation state component, *Types*. This set has relations over it which act to constrain possible values of the relations over activities in models of the theory - these are described in the fourth part of the chapter.

An index of state components, the sections in which they are discussed, and the classes in which they are declared is included at the end of Chapter 10, on page 168.

During the development of the theory, the role of formal proof has been fairly small: the intellectual effort in the PhD has gone into the representation of the domain as a formal theory. Consistency checking has been carried out throughout the project, but more by way of inspection and reasoning than formal syntax manipulation. In general, if a particular property was desired of the theory that property was expressed as an axiom, or invariant, of the theory rather than being formally derived from others. Once expressed as an invariant the property could be assessed for consistency with other invariants using the informal mechanisms of inspection and reasoning. Nevertheless, some consistency proofs are presented below, and some theorems derived from the invariants shown along with their derivations.

In the following description of the theory, type declarations which are as they appear in the theory are labelled T*n*, and invariants I*n* (where *n* is a number). This is to facilitate the discussion of invariants that have been introduced earlier, and to help the reader find them in the formal presentation of the theory given in Appendix 2. If a type declaration or invariant label is suffixed with a prime ('), it is a correct but partial representation of an invariant from the theory. Invariants and type declarations introduced without a number come from early incorrect versions of the theory indicating that a refutation of the property they enforce will be given prior to the introduction of their replacements.

Although the notation has been briefly described in an earlier chapter, many of the underlying concepts are discussed as they are introduced to aid readability, as are additional explanatory devices, terminology and different forms of theory and model representation.

9.2 *Activities* and Its Subsets.

9.2.1 Introduction

This section describes the most fundamental concept in the domain theory - the medical activity. This is represented formally by the set *Activities*, and partitions of that set - *Request*, *Proceed* and *Complete*. In models of the theory, *activities* have a lifecycle, being created as members of *Request*, and subsequently

moving between the partitions in a controlled manner, eventually to be a member of the set *Complete* whereupon no further change is permitted. Operations that enable these state changes are described and discussed below. The need to limit the organisational scope of specialisations of the theory is explained, and a crude mechanism for achieving this is described.

9.2.2 *Activities*

An activity is the term used to talk about a medically meaningful encounter between a patient and some attribute of medical care. The word attribute is used as although we might immediately imagine an activity as concerning a patient and a health care professional, there are more abstract activities such as GP care which continue over long periods of time, but are still activities in the sense of the theory. An activity may not have started but still exist - a booking for an operation for example - or it may be currently in progress, or it might have already been completed.

This is the intended interpretation: the theory merely represents *Activities* as a set of a given, though unspecified, type. Thus we say that

T_1: *Activities*: *Set*[A]

which means that we define the label *Activities* to mean a set, all of whose members are represented in the set A. The notation actually says that the *Activities* is a member of the 'Power Set' (the set of all subsets) of A. (Other notations use the symbol $\mathcal{P}$ for power set in place of Set[]).

Throughout this presentation, large Sans-Serif type is used to represent these unspecified 'carrier sets' that define the type of the most basic concepts. The existence of carrier sets reflects the notation's compliance with the notion of 'strong typing' which is a characteristic of the version of set theory on which the notation is based [Hayden68]. Strong typing outlaws certain (algebraic) operations on different types. For example we cannot construct the union of two sets that have different underlying types as defined by the carrier sets from which they are derived.

9.2.3 *Request*, *Proceed* and *Complete*

Activities have a lifecycle, and may exist in any of a number of stages. An activity may not yet exist whence it is in the set difference of A and *Activities*. An activity may have been requested but not be currently in progress or have finished (maybe it has not yet started) whence it is a member of the set *Request*, or it may be currently in progress whence it is a member of the set *Proceed*, or it may be finished whence it is a member of the set *Complete*.

A member of the set *Request* is also an activity, so we will want to say, or be able to derive

Request $\subseteq$ *Activities*

If the above is true, we know that *Request* must also be a subset of A, and so a member of Set[A]. In other words that

T_1: *Request*: *Set*[A].

The same is true for *Proceed* and *Complete*, so the most basic type definition we have is

T_1: *Activities*, *Request*, *Proceed*, *Complete*: *Set*[A].

The commas separating the set names mean that they all are of the same type. The initial type declaration tells us what sort of a thing is represented by a particular label: all the labels in the previous type definition refer to subsets of A.

9.2.4 A Word on Type Declarations

The notation allows the definition of types in terms of constructed quantities, providing all of those quantities have themselves been defined earlier. Thus later on we say that ActAtt (the set of all activities currently attended by a patient) is defined as

T6: *ActAtt*: *Set*[*Activities**Complete*].

This is possible in the notation as by the time we want to define ActAtt, we have already defined *Activities* and *Complete* (insofar as a 'previous' class has already defined them in terms of a carrier set). The use of previously defined quantities in a type declaration is a form of shorthand, bringing together a type declaration based on carrier sets with an invariant rule. For example the previous type declaration might have been written

ActAtt: *Set*[A]	Type Declaration
ActAtt $\subseteq$ *Activities**Complete*	Invariant

9.2.5 Invariants over *Activities* and its Subsets

The type definition tells us very little about the nature and behaviour of the concepts defined. This is expressed using invariant properties linking different declared quantities together. In the case of the sets *Request*, *Proceed* and *Complete* we know these are all subsets of *Activities*. We also want to say that an activity (that exists - not some hypothesised future activity) must be in one of these sets. These are invariant properties in that they always hold over all valid models of the theory - no matter what, all members of *Request* are also members of *Activities*. We express both of these invariants in one expression:

I1: *Request* $\cup$ *Proceed* $\cup$ *Complete* = *Activities*.

The union of the three sets *Request*, *Proceed* and *Complete* is equal to the set *Activities*. From this we can deduce, using set theoretic axioms, that *Request* is a subset of *Activities*.

We know from

Request $\cup$ *Proceed* $\cup$ *Complete* = *Activities*

that

Request $\subseteq$ *Activities*

We also want to say that not only must an activity be in one of the sets *Request*, *Proceed* or *Complete*, but it must be in no more than one: an activity cannot be proceeding and have been completed simultaneously (or at least, this is an axiom that we did not manage to refute, or indeed expect to). We say this in the following invariant:

I2: *Request* $\cap$ *Proceed* = *Proceed* $\cap$ *Complete* = *Complete* $\cap$ *Request* = $\varnothing$.

None of *Request*, *Proceed* or *Complete* intersect with any other, or at least their intersections contain no members.

The major difference between the type declaration and the invariant is that the declaration is definitional and says what sort of thing the state component is while the invariant is an axiom that describes limitations to and constraints on its behaviour. The important distinction as far as we are concerned is that the invariants are assertions that are waiting to be refuted. For example, it seems pretty clear that an activity is never both a *Request* and *Complete* at the same time, but this is a claim that the analyst has made about the nature of the domain. If an instance of an activity that was both a *Request* and also *Complete* could be found (though what sort of a thing that would be is difficult to envisage), then the theory would be invalid and would have to be re-worked. A refutation of the axioms expressed so far would be pretty incredible, but for some of the more complex invariants described later, counter-examples are easier to imagine, and in some cases were found.

9.2.6 The First Class Schema

We now have our first class definition schema

ActClassOld

$$\begin{array}{|l} Activities, Request, Proceed, Complete: Set[A] \\ Request \cup Proceed \cup Complete = Activities \\ Request \cap Proceed = Proceed \cap Complete = Complete \cap Request = \varnothing \\ \hline\hline Activities' = \varnothing \\ \hline \end{array}$$

The name of this class, or theory fragment, is ActClassOld. 'Old' is suffixed to indicate that this was not the final version.

The initialisation pseudo-event gives the set *Activities* a starting state of the empty set in all models of this class. Because all the other state components defined here are subsets of *Activities*, their initialisation values are given also as

$X \subseteq \varnothing \Rightarrow X = \varnothing$ for all X.

At all stages of the construction of the theory, we ought to construct and discharge proofs that the resultant set theoretic expressions, declarations, invariants and other rules are at least consistent in the universe of set theory. If this is not the case then the theory is not valid and cannot represent anything in the world. As explained earlier, formal proof did not play a major part in the development of the theory, the author relying on inspection and reasoning instead. Informally we can readily see that this class can be implemented consistently and non-trivially.

The proof obligation for a state schema is class consistency. That is, there is an object which obeys the rules of set theory and the invariants of the class. There is a subsidiary proof obligation which is to show that the value of a model (or object) after initialisation is a valid state for a model of that class. Clearly the demonstration of the existence of a valid initialised object also discharges the first proof obligation. In the case of this initial state schema, the initialisation 'post-condition' is

$Activities' = \varnothing$.

Now we must see if there is a possible model of this class where its state components are of such a state that *Activities* is the null set and all the invariants are satisfied. Such a model has the following instantiated state components:

$Activities = Requests = Proceed = Complete = \varnothing$.

The first invariant is

$Request \cup Proceed \cup Complete = Activities$

substituting the values of the sets in the initialised model for the state component names in the above expression we get

$\varnothing \cup \varnothing \cup \varnothing = \varnothing$

which is true: this model complies with the first invariant. The second invariant is

$Request \cap Proceed = Proceed \cap Complete = Complete \cap Request = \varnothing$.

Again, substituting the model values for the names above we get

$\varnothing \cap \varnothing = \varnothing \cap \varnothing = \varnothing \cap \varnothing = \varnothing$

which is again true. We have thus demonstrated that a valid model of this class exists.

9.2.7 The Operations: First Version

It is all very well to know what states the system might possibly exist in, but we have not seen how those states might be reached. In other words we know something about the structure of possible models of the theory, but not much about their behaviour. This is given through the definition of operations that might be invoked that change the state of the system.

We know that all models of the class ActClassOld start off with all their state components set to equal the empty set (by virtue of the initialisation pseudo-event), so their are no activities at all before we invoke an operation on a model (or object) of the class.

The operation that is responsible for the creation of the activity in this class is called

ActClassOld.Request($\rightarrow a$)

This operation takes no argument - we can invoke it without stating what values it is to use in its operation - however, it does return an argument: a. This is indicated by the arrow in the brackets following the operation name. The arrow is merely a decoration to aid our interpretation of the operation - it has no formal semantics, and when considering pre- and post-conditions, as well as during the static analysis of the class we must ignore it.

The argument a must be described in the operation schema as we do not know anything about it so far. Note that a is not a value, or an element of any set, but is a variable name for which can be substituted real values or elements from sets when we investigate behaviours of models of the system. The role of arguments in operations is much the same as variables in familiar school algebra: compare with, for example, x in *'Let x be the unknown, and suppose* $x^2 + y^2 = 4$*'*.

We must give a type for the returned argument, and say what the state of the system in relation to the value of the argument must be if the operation is to be allowed. This is the pre-condition of the operation. In the case of the operation *.Request* (I will use this shortened form where it is not confusing to do so - the "." distinguishes the operation *.Request* from the state component *Request*) the type of the argument is given by the expression:

a: A

or a is of the type A - it is a member of the set A. and the precondition by

$a \notin Activities$

or a is not a member of the set *Activities*. We can express this more succinctly as:

a: A*Activities*

or a is a member of the set difference of A and *Activities* - it is in A but not in *Activities*.

We know when the operation might be invoked - whenever a is in A but not in *Activities* (although the intended interpretation is that the invoker has no control over the value of a, as explained the arrow in the operation name has no formal semantics and so this is technically a precondition) - we need to know what happens as a result of said invocation (or else our behavioural description would not have achieved much). This is given in the post-condition:

$a \in Request'$.

Where the dash after the name of the state component indicates that the predicate applies to its value after the invocation of the operation. What the post-condition says is that following the operation, a is a member of the model of the state component *Request*.

This operation can be expressed in the operation schema:

ActClassOld.Request$(\rightarrow a)$

a: A*Activities*
$a \in Request'$

.

Another operation that we want is that which starts the request and turns it into a proceeding activity. The operation that achieves this is

ActClassOld.Start(a).

For this operation the invoker must provide the argument herself: we must 'tell' the model which activity it is that we want to start. In the pre-condition for this activity we want to say that we can only start activities that already exist, and moreover have not yet been started and are still requests. This we do by asserting:

a: *Request*.

Subsequent to the operation we wish to say that the activity has started - it is a member of the set *Proceed*. Thus:

$a \in Proceed'$

and the operation schema is

ActClassOld.Start(a)

a: *Request*
$a \in Proceed'$

.

In the same way as the static class schema, we can informally demonstrate the consistency of this operation schema.

We must discharge two proof obligations for the operation schema. The first is to show that there is a state of a model of the class where the precondition and the invariant allow the operation to occur. The second is to show that whenever the operation is invoked, there is a possible model such that the postcondition and invariant can both be observed. We can (informally) discharge the first proof obligation by proposing the following model:

$Activities = Request = \{a1\}$
$Proceed = Complete = \varnothing$

Here the type declaration counts as a sort of pre-condition - there must be at least one element in *Request* for us to be able to extract. This can be shown to satisfy the invariant properties of the class through substitution as follows:

$\{a1\} \cup \varnothing \cup \varnothing = \{a1\}$

which is true, and

$\{a1\} \cap \varnothing = \varnothing \cap \varnothing = \varnothing \cap \varnothing = \varnothing$

which is also true. We can discharge the second proof obligation by showing how to construct a state of a model after the operation for any value of that model where the operation is legal. One such construction is as follows:

$Activities' = Activities$
$Request' = Request \setminus \{a\}$
$Proceed' = Proceed \cup \{a\}$
$Complete' = Complete$

where a is some element of *Request* - in fact the one indicated by the argument of the operation (we know there must be at least one such element from the implicit precondition in the type declaration).

We know that the post-condition to the operation is satisfied as

$a \in Proceed \cup \{a\} = Proceed'$

whatever the value of *Proceed*. We can show that the invariant is satisfied by replacing the dashed state component names with the undashed ones according to the equalities above.

We want to show firstly that

$Request' \cup Proceed' \cup Complete' = Activities'$

but we know that the left hand side of this expression can be re-written as

$(Request \setminus \{a\}) \cup (Proceed \cup \{a\}) \cup Complete \equiv$
$((Request \setminus \{a\}) \cup (Proceed \cup \{a\})) \cup Complete \equiv$
$((Request \cup (Proceed \cup \{a\})) \setminus (\{a\} \setminus (Proceed \cup \{a\}))) \cup Complete \equiv$
$((Request \cup (Proceed \cup \{a\})) \setminus \varnothing) \cup Complete \equiv$
$Request \cup \{a\} \cup Proceed \cup Complete \equiv$
$Request \cup Proceed \cup Complete$

as we know that a is an element of *Request*. Now we know from the invariant that

$Request \cup Proceed \cup Complete = Activities,$

and as

$Activities' = Activities$

we can see that the invariant with dashed state component names holds. For the second invariant we want to show that

$Request' \cap Proceed' = Proceed' \cap Complete' = Complete' \cap Request' = \varnothing.$

Substituting the undashed state component names we get

$(Request \setminus \{a\}) \cap (Proceed \cup \{a\}) = (Proceed \cup \{a\}) \cap Complete = Complete \cap (Request \setminus \{a\}).$

Taking the first expression (before the first equality)

$(Request \setminus \{a\}) \cap (Proceed \cup \{a\}) \equiv$
$(Proceed \cup \{a\}) \cap Request \setminus \{a\} \equiv$
$(Proceed \cap Request) \cup (Request \cap \{a\}) \setminus \{a\} \equiv$
$\varnothing \cup \{a\} \setminus \{a\} \equiv$
$\varnothing$

as we know that a is an element of *Request*, and *Request* and *Proceed* are disjoint. For the second expression we have

$(Proceed \cup \{a\}) \cap Complete \equiv$
$(Proceed \cap Complete) \cup (\{a\} \cap Complete) \equiv$
$\varnothing \cup (\{a\} \cap Complete) \equiv$
$\varnothing \cup \varnothing \equiv$
$\varnothing$

as a cannot be a member of both *Request* and *Complete* due to their disjunction. Finally the third expression gives us

$$Complete \cap (Request \setminus \{a\}) \equiv$$
$$(Complete \cap Request) \setminus \{a\} \equiv$$
$$\varnothing \setminus \{a\} \equiv$$
$$\varnothing$$

as *Complete* and *Request* are disjoint. We have thus shown that from the invariants, when the state of the model is such that the pre-condition to the operation holds, we can always construct a state of the model such that the post-condition and the invariants hold.

The above proofs are sufficient (albeit informally) to show the applicability and effectiveness of this operation. Proofs of class consistency and operation applicability and effectiveness are not discharged elsewhere in the thesis due to the length of their exposition.

We have seen an operation that creates an activity and one that starts a request. Other reasonable operations that are defined in this class are *.Complete* which takes a member of *Proceed* and puts it in the set *Complete*, and *.Cancel* which takes a member of *Request* and removes it from sight forever. When a patient has been booked in to attend the clinic, but then cancels there appointment, perhaps because they have moved away or their GP decides the referral is inappropriate after all, or the patient has died, we can forget all about the activity - it is of no consequence to the clinic and does not effect how it functions in future (though we might like to keep a record of it) - it is as if it never existed which in a sense is true. An activity that has started cannot be so denied, nor one that has finished. This is the interpretation of the operation *.Cancel*: it can only be applied to these 'fictitious' activities that have never started: the pre-condition insists that 'a' is a member of the set *Request*.

In this class, we can express the behaviour of individual activities using a state-transition diagram. This is not generally a feasible way of understanding class behaviour, but as ActClassOld is so simple structurally and dynamically, and because one activity behaves totally independently of all others we find that such a diagram is indeed a useful tool. A state transition diagram for a member of the set A is given below.

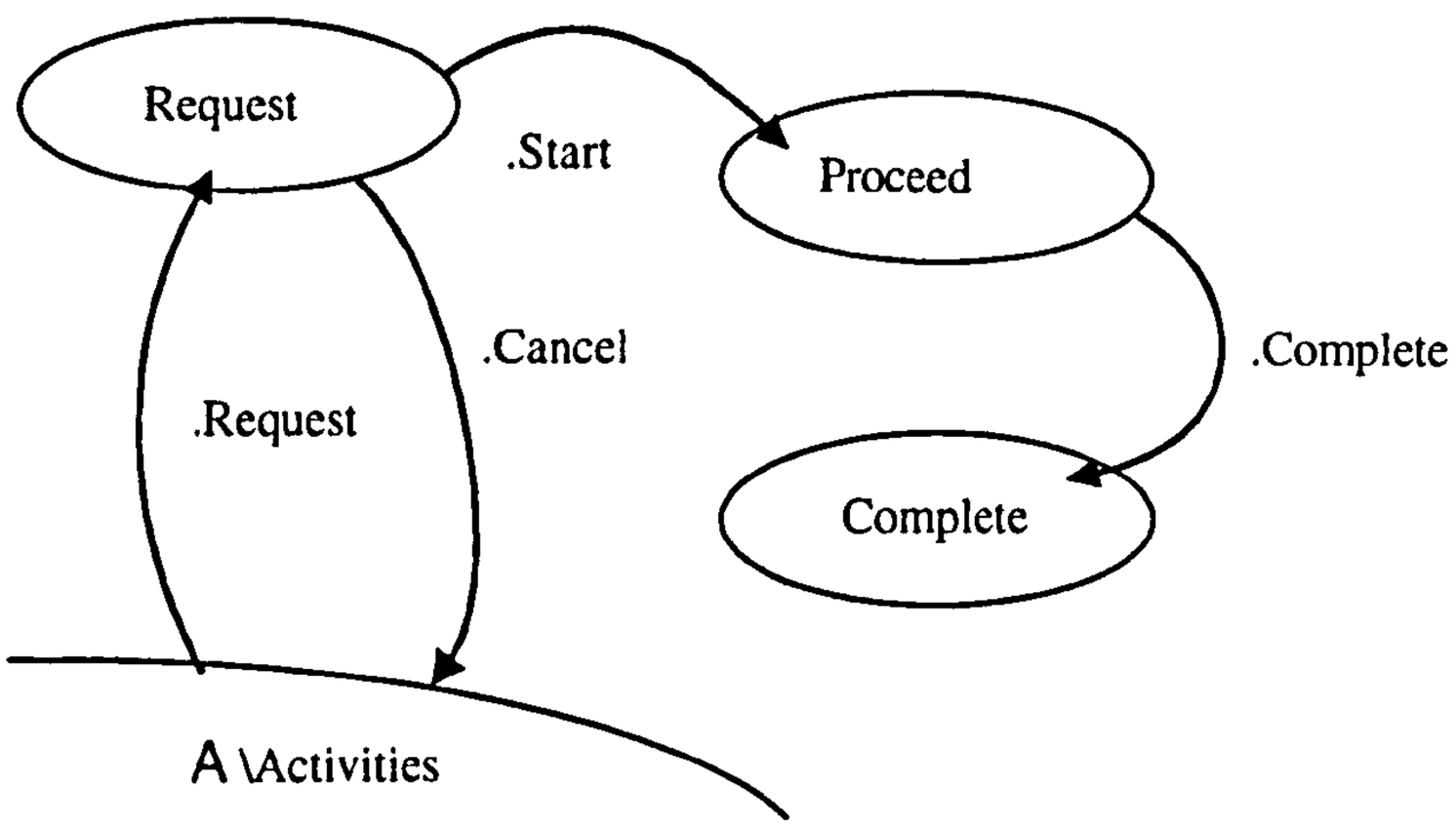


Figure 3-6: State transition diagram showing the possible operations in the class ActClassOld on a member of the set A, depending on the state of that member.

This state transition diagram shows that there is a standard lifecycle for activities - if an activity is not cancelled (in which case it is as if it has been destroyed and we are no longer concerned with it) then it moves from *Request* to *Proceed* to *Complete*, always in that order, when it is completed, nothing can change its state.

9.2.8 The Operations: Second Version

The class as it stands is an extremely basic theory of how a clinical domain behaves. Of course there is much more to a clinic than the set of *Activities* (we have not spoken formally even about patients yet) just as there is more to a hot cup of tea than the theory of thermodynamics. Although the scope of the theory so far is limited, we would hope that what little it does have to say by way of predicting static and behavioural properties of the domain is accurate. Newton's laws of mechanics do not give us a total understanding the solar system, but the predictions they do enable us to make are valid (more or less). The way the laws given by the theory we are developing work is as a set of constraints that forbid certain model states and behaviours. An activity is forbidden to be both a *Request* and be *Complete* at the same instant. An activity that is in *Proceed* and thus is currently in progress is forbidden from becoming a *Request* or ceasing to exist altogether: it may become a *Complete* activity, but equally it may not and stay as in *Proceed* forever. If we can find real examples of activities that disobey these laws, then the theory is invalid and must be re-worked. Even at this early stage of the analysis we did indeed find such counter examples that 'refuted' the behavioural laws given in ActClassOld.

The theory as it stands forbids an activity to be 'created' as a member of the set *Proceed*: it must first become a *Request*. This is fine for the bulk of clinical activity, but even in a mainly outpatient directorate such as the Endocrine directorate, emergency work is extremely important. When a patient is delivered by ambulance to the Accident and Emergency department, treatment starts immediately - the activity never passes through the *Request* state. Although an emergency patient is passed to the Endocrine directorate through a process of service request, the theory is supposed to be sufficiently general to cover all domains of medical care and so should be able to address the emergency as seen from the Accident and Emergency department. Not only do emergencies directly achieve the *Proceed* state, but in the day centre, telephone calls to the doctor or, as happens more frequently, to the specialist nurse are not first requested and then started - the activity starts as soon as the telephone is picked up. There is no operation in the old version of ActClassOld that allows for this creation and commencement simultaneously: the theory forbids it and has thus been refuted. A more realistic theory would accommodate such a possibility, and indeed a new theory was constructed which included an operation called **ActClass1.*SuddenStart***(→a) which takes a member of A*Activities* and puts it into the set *Proceed*.

The original theory expressed by the class ActClassOld allowed for activities in *Proceed* to stay in *Proceed* or move to *Complete*: interruptions were forbidden. Early on in the analysis process, it became apparent that not only did interruptions regularly take place, but many of them were important for patient care. Particular findings regarding interruptions recorded at interviews with a clinicians were:

- Drs frequently have the activities they are involved in interrupted. They might be called / bleeped by a junior Dr about one of the inpatients they are responsible for. They might be telephoned by a patient's GP or community nurse for some advice about a particular patient.

- Interruptions are significant for patient - after all, there must be a good reason for interrupting a Dr.

- Interruptions can often be thought of as mini-activities (some with the patient not present).

It seems that another of the behavioural predictions of the theory - namely that an activity that is a member of *Proceed* can only be moved to *Complete* - has been refuted.

If an activity has been interrupted, it cannot continue to be in the set *Proceed*. A new theory would have to address this, either by introducing a new disjoint subset of *Activities* (perhaps called 'Interrupted'), or create a new operation that moved the activity from *Proceed* back to *Request*, whereupon it could be (re-) started at any time. The latter solution was chosen as it introduced minimum confusion and complexity to what was the most basic class in the theory. The new operation is called **ActClass1.Suspend**(a). There is no limit to the number of times an activity might be started and suspended - it can still only be completed once.

The behaviour of activities in the new class can be represented in the following state transition diagram.

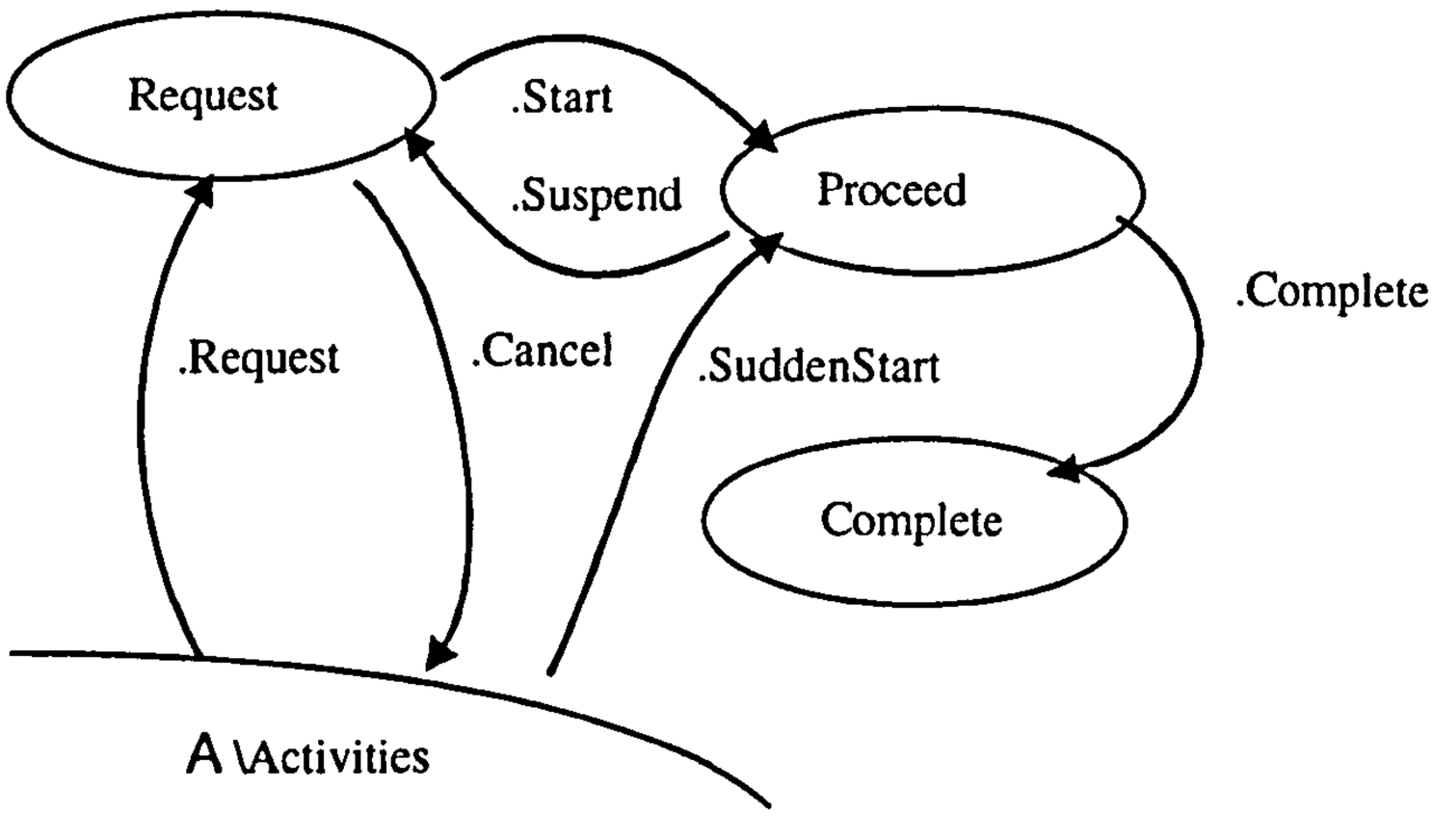


Figure 3-7: State transition diagram showing the possible operations in the class ActClass1 on a member of the set A, depending on the state of that member

One of the features of the new class is that it permits models in which the following sequence of events is observed. An activity is requested whereupon it becomes a *Request*, it is then started and becomes a member of *Proceed*. Something happens to cause the clinician to suspend the activity at which point it is returned to *Request*, and as the activity is now a member of *Request*, it can be cancelled and the clinic will behave as if it had never existed. This is nonsense: we cannot deny the existence of an activity just because it has been suspended. In this case the anomaly would be fairly easy to fix: an introduction of a new subset of *Activities* as was proposed earlier would enable us to prevent such an undesired sequence of operations.

Although the theory as it currently stands allows for this unrealistic behaviour, we have already seen that its strength lies in its prohibitions rather than its exhortations. We can fail to discover behaviours that are allowed by the theory and still have reason to believe in its validity. If we discover behaviours in the domain that are disallowed by the theory, then the theory has been refuted, and must be discarded. Thus the theory as it stands at the moment might not be as 'good' or 'realistic' as one in which a suspended activity could not be cancelled, but it is not 'wrong' insofar as it has not been refuted.

Of course if we are not bold in our attempts, and try merely to create a theory that is not refuted, we might end up with one such as has been described. The purpose of this analysis is to understand and represent the domain into which an information system might be placed. The theory we have developed so far,

concerning only state changes of activities, does not really tell us anything useful at all: it lacks semantic richness. The more wealth of detail we add, providing the theory is not refuted, the more aspects of the domain we will have represented. The 'scientific' method that was used in the development and refinement of the theory does not help us to decide which aspects of the domain should be modelled (for this we must resort to common sense and guidelines such as those discussed in Section 7.3), it merely helps us to see if we have made an error in the representation of the domain.

We have seen that the theory as it stands so far is not 'wrong' in the scientific (as we have defined it) sense, but contains insufficiently rich semantics reflected by a paucity of state components (patients have not been mentioned yet) and by its failure to prohibit behaviours of models that are never seen in the domain (as seen in the possible cancellation of suspended activities). Many state components have been introduced into the theory as we shall see, addressing the problem of insufficient state components. As for the 'sloppiness' of the behavioural specification of the theory so far, we shall see that cancellation of interrupted activities is forbidden later on after time has been introduced: we will see that only activities that have never been started can be cancelled.

9.2.9 Limiting the Scope: Introducing Boundaries

Although we want the theory to be designed such that it can be specialised to any area of medical activity, we do not need to (and could not hope to) represent the totality of medical care. For any specialisation there is thus a boundary between those activities that are pertinent to the medical domain we are interested in and those that are not. The theory distinguishes those activities that are inside the boundary - whereby they are members of the set *In* - and those that are outside the boundary - which are members of *Out*. The intended interpretation of these concepts is as follows. Any activity that the medical domain we are currently concerned with is responsible for carrying out is a member of *In*, and all those that we are interested in but have no control over are members of *Out*. In the case of the DEDC activities of type Doctor Consultation and Diabetes Care would be members of the *In* set whereas those of type Blood Test (carried out by Clinical Chemistry) and Surgical Intervention (carried out by Surgery) would be in *Out*.

The partitioning of *Activities* into *In* and *Out* is 'orthogonal' to that into *Request*, *Proceed* and *Complete* in that a member of either *In* or *Out* can be in any of the temporal subsets. We are not so concerned with the details of *Out* activities so the operations on these are simpler: ***.OutRequest***(→a) which produces a member of both *Request* and *Out*; ***.OutProceed***(a) that moves an external activity from *Request* to *Proceed*; and ***.OutComplete***(a) that moves an activity to the set *Complete*. We are not interested in whether or not an external activity has been interrupted, so there is no equivalent to the ***.Suspend*** operation, and the ***.Cancel***(a) operation can be applied to activities that are members of either *In* or *Out* (as long as they are also in *Request*).

9.2.10 Conclusion

In this section a number of very simple, but very important, operational state components from the domain theory have been described. Even this early in the presentation of the theory, we have seen how some behavioural properties of the theory were refuted: specifically the first version of the theory presented did not allow for an un-requested activity to start (as happens in emergency and unplanned activities), or for an activity to be suspended. The two new operations ***.SuddenStart*** and ***.Suspend*** address these problems, but have the side effect of admitting the unrealistic case of an activity being requested, started, suspended then cancelled. This 'semantic sloppiness' is not refuted but goes against the spirit of the first of the scientific imperatives we are observing - that we should strive for as much realism and

boldness in our theories as we can. This particular problem is solved when time is introduced in a subsequent class.

The partitions of *Activities*, *In* and *Out* are introduced which are a simple means of creating a boundary of our 'domain of interest': activities that are in *In* are inside the boundary; those in *Out* are outside. There are no operations which change which of these partitions an activity is in.

9.3 Graphs Over *Activities*

9.3.1 Introduction

As it stands, the theory, although 'correct' inasmuch as it has not been refuted, is not particularly illuminating. The first of our two scientific imperatives exhorts us to add more richness and constraint. So far, *activities* are described as existing in isolation - in fact medical care consists of complex interactions of aggregates of medical activities. One of the thrusts of this work has been to find how these aggregates are constructed and structured. Observation of medical care as a discipline (and many if not most other disciplines besides) seems to indicate that some activities are before others, and some activities are a part of others. In the theory these properties of medicine are represented as relations over *Activities*. A relation where the domain and codomain are of the same type (ie they are subsets of the same carrier set) is called a graph. Where the relation is also a function, the graph is also known as a tree. A useful metaphor to use when talking about graphs and trees is that of a family tree. This metaphor is briefly discussed below and is used subsequently when discussing appropriate aspects of the theory.

9.3.2 *Before* and *After*: Medical Ordering Introduced

Before and *After* are both graphs over activities which represent aspects of their chronological ordering. As explained earlier, these concepts have not been used greatly in the specialisation of the theory to the DEDC (they are only used in the definition of a followup activity). They have been left in as it was felt that they might be useful in other areas of medical care, specifically to support the notions of Care Profiles and Care Protocols. The way in which the interpretation of the relations changed through the refutative process is interesting, so they are discussed below.

The types of these state components are given as follows:

T2. $Before, After: Activities \leftrightarrow Activities$.

Both are partial relations - we do not want to be forced to order all activities with respect to each other, and indeed in many cases we can not. The chronological nature of the relations is given in the invariants:

I3: $Before = After^{-1}$

I4: $After^{+} \cap id[Activities] = \varnothing$

I5: $(im\ After)\ (Proceed \cup Complete) \subseteq Complete$.

Before and *After* are the inverse of each other: in other words if activity $a1$ is *Before* activity $a2$, then we know that $a2$ is *After* $a1$. The second predicate says that the graph *After* (and hence the graph *Before*) is acyclic: if we arrange activities in a sequence such that every activity is *After* the preceding activity in the sequence, then we will never find the same activity at two locations in the sequence. An activity can never be *After* itself, or *After* an activity that is *After* itself, and so on. The third invariant says that any activity

that a member of *Proceed* or *Complete* is after must itself be complete. If we consider an activity $a1$, then all activities *Before* $a1$ must be completed, and hence members of *Complete*, if $a1$ is to start.

These invariants between provide constraints on possible values of *Before* and *After* that we would expect to hold in any conventional understanding of the nature of time. This does not tell us anything particularly interesting: if one activity takes place before another we should not be surprised to see it finished before the second commences. These invariants do not say anything about when we might want to say that one activity was after another: we must be careful how we decide on and explain the intended interpretation of the concepts.

We are interested for now in structural and behavioural properties that are essential to an understanding of the medical domain. There is nothing essential to medicine about the observation that 4 O'clock in the afternoon is after 3 O'clock in the afternoon: in other words we do not want a representation of time (yet). The sort of thing that is important to medicine is the observation that Mrs Jones was admitted for surgery after she had a series of outpatient encounters with a consultant from the hospital. Two activities related through *After* must be so ordered for some reason germane to the delivery of care.

Similarly, if we want to the state components to have some effect on the behaviour of models of the theory as a whole then *Before* and *After* must say something about the domain as it is unfolding, not just record some subset of the observed temporal sequencing of activities that have taken place. Thus if an activity $a1$ is *Before* $a2$, part of the interpretation is that $a1$ must have finished before $a2$ can start, even if at this stage both activities are members of the set *Request*. In short *Before* and *After* represent behavioural imperatives and not just historical records.

The precise interpretation that was desirable is still not totally clear, and indeed changed over the course of the theory's development. The nature of that change is interesting, and is presented below. The following argument makes use of state components that have not yet been introduced: these are explained here but should not cause concern, what is important is how the interpretation of *Before* and *After* changed as a result of the discovery of a counter-example that refuted a theorem of the theory (expressed as an invariant).

9.3.3 The Problem of the Blood Test

One of the theorems that was included (as an invariant) in an early version of the theory related possible orderings of activities with participants involved in those activities. The form of the theorem was:

$$\forall aa: After^{+} \cap (Dom(Participant))^2 \bullet (Participant \circ \{aa\} \circ Participant^{-1}) \cap id[NSR] \neq \varnothing$$

At this stage of the development of the theory, the nomenclature of the various sets and relations was as follows. *NSR* is the set of 'Non Shareable Resources', a fairly unpalatable name for clinicians, patients, and other resources that can only be in one place at a time. This is in contrast to consumables such as 'bandages' or 'swabs' that by their continuous nature can be used in a number of sites simultaneously (at least they are considered to be continuous - we are not concerned with the identity of individual swabs and bandages). Participant is the relation that links *NSR* with *Activities*. Thus (*im Participant*) $\{a\}$ is the set of clinicians, patients and resources that are currently physically present at activity a.

What the invariant says then is that any two activities which have clinicians, patients or resources physically associated with them and that are ordered with respect to each other must share at least one of those associated 'non-shareable resources'. This seems to make sense - after all why would one activity

have to be after another unless it was because the patient involved, one of the clinicians, a room, or a piece of equipment was needed for both, and could only be used by one activity at a time? Although the explanation of the invariant is fairly long-winded, its statement in the set-theoretic notation above is succinct and precise, and predicts a definite property of the organisational system.

The theorem survived for two iterations of the theory, but was 'refuted' by the example of the blood test. It is necessary that the blood for a blood test is taken before the blood is analysed, but none of the same clinicians, resources or the patient need be present at both the extraction of the blood and its analysis. Thus it seemed that the theorem as it stood did not accord with the observed facts.

A number of ways in which the theory could be adapted to accommodate this problem were considered. Firstly the blood sample could be designated a participant of the activity. This seemed a little bizarre, and did not really fit in with any intuition of what sort of thing a participant in an activity was. Alternatively, Parts could refer to the patient the activity was about, not just the patient who was present at the activity. This would do, but could be viewed as the addition of an exception solely to allow the invariant to be preserved rather than add any insight into the nature of the domain. The third approach considered which was adopted was to re-define *After* and *Before* so as to be 'medically meaningful'. The ordering of two operations in an operating theatre is necessary for scheduling reasons. The ordering of a test and a visit to the doctor to review that test is considered necessary for medical reasons. Thus the theorem became:

$$I_{14}: ActSubject \circ (Includes \cup After) \circ ActSubject^{-1} \subseteq id[Patients]$$

This says that any two activities that are related via the *After* or *Includes* (see below) relation must refer to the same patient (via the function *ActSubject* which is described in more detail below). In this case, the refutation of a theorem led to a change not only in the structure of the theory, but also a re-interpretation of some of its components.

The concepts *Before* and *After* are in fact more complex still as it is almost never the case that one medical activity must take place after another: if a blood test is required before a doctor sees a patient, but for one reason or another has not been carried out, the doctor's consultation with the patient will still take place but will be of a lesser quality. The assumption embedded in the theory that applies here is that in order for the later activity to start, the earlier activity must either be completed, or cancelled in which case the pair is removed from the *After* (and *Before*) relation: this is a reasonable theorem, but might not accurately represent what actually happens. The relations are used in other more subtle ways throughout the exposition of the theory however, and it is imagined that they would come in useful if the concept of care plans was to be described.

9.3.4 *Includes* and *During*

In the early stages of the theory construction process, it seemed that the concepts *Before* and *After* would be extremely useful - after all what is medical care other than a time ordered sequence of clinical interventions of one sort or another? It was found over the course of the work that although looking back on a medical history, one can re-construct such a sequence (or at least imagine that one has done so), the use of such ordering to constrain the process of care as it takes place is too prescriptive: the business of medicine is not only extraordinarily unpredictable, but the exceptions are of great importance and so must be accommodated in any theory of care (the Care Profiles and Care Protocols mentioned earlier in this respect are precisely rules and guidelines laid down to ensure consistent good quality care). Of much greater use in the development of the theory was the notion of activity inclusion. This was considered of

potential though secondary interest at the start of the analysis, yet was developed to be the main form of structuring used.

The intended interpretation of *During* and *Includes* is, as with *Before* and *After*, a medically meaningful one. If activity $a1$ *Includes* activities $a2$ and $a3$, then activities $a2$ and $a3$ are medical components of $a1$: in attempting to deliver the care implied in $a1$, activities $a2$ and $a3$ are created and discharged. We can see how this might work in the case of a blood test. Before a blood test can be satisfactorily completed, a blood sample must be taken, and that sample analysed. If we call an instance of a blood test $a1$, and the sampling and analysis of the blood $a2$ and $a3$ respectively, we can see that $a1$ *Includes* both $a2$ and $a3$.

As with *Before* and *After*, *During* and *Includes* are graphs and are the inverse of each other. We could say that

T3: $During, Includes: Activities \leftrightarrow\!\!\!\!\!+\!\!\!\!\!\rightarrow Activities$

and

I7: $During = Includes^{-1}$.

For much of the theory's development it was considered that *During* was not only a partial relation, but a partial function also (also called a tree when it is a subset of a graph as in this case):

$During: Activities \rightarrow\!\!\!\!\!+\!\!\!\!\!\rightarrow Activities.$

In other words, any activity can be a part of at most one other activity. Thus a dietitian consultation is a part of one instance of dietitian care and not of any others, and a particular blood test might be a part of doctor care (although other health care professionals may want to look at the results, this is through the sharing of information - the blood test is still a 'part of' the health care process that it was requested to support, in this case an instance of doctor care). The assumption that an activity could be a part of at most one other activity, as recorded through the representation of the *During* relation as a tree, was taken early on in the project, and was not questioned for many iterations of the theory development cycle. This is an example of a 'paradigm trap' such as is discussed in Section 13.3. It is also useful here to record that refutative examples of behaviours of the domain do not 'leap out' at the analyst, but must be sought after. The good analyst, much as the good scientist, is beholden to try and refute his or her theory as determinedly as possible: it is not acceptable to create a poor theory and then act as an apologist for it. The intention of the author was to act as a responsible analyst - the success and sincerity of the effort is for the reader to decide.

The assertion that *During* behaved as a partial function was refuted however, and many counter-examples have been found since then. The example used in the original refutation was that of a nurse changing a bed-pan once for a patient who was in hospital for two completely different operations. Changing a bed-pan might be a required activity in the 'care plans' for both concurrent episodes of care, but we would see it take place in the domain only once: it is a component of both clinical procedures. It would be possible to construct a type structure that could support this problem, and prevent it from contravening the invariants (changing the bed-pan, for example, might be thought of as a direct part of some high level 'in patient episode' activity or some other artificial activity type), but there are other aspects of care that still cause problems. One of these is observed in the care delivered to diabetic women who become pregnant. The diabetic woman will typically be a long-standing patient of the day centre (unless the condition is Gestational Diabetes Mellitus - diabetes confined to pregnancy) and will have had many activities

completed of which she was the subject, under a single instance of 'diabetes care'. Many of the activity types she will attend as part of the care delivered by the hospital during the ante natal and confinement stages of the pregnancy are common to all such 'patients', but some are provided for only those pregnant women who have diabetes. An example of such an activity type is the combined clinic run by the Obstetrics and Gynaecology Directorate in association with the Endocrinology Directorate. The activities that take place within this clinic are in some sense, and at some level, components of both Obstetrics & Gynaecology care and Diabètes Care. Multiple conditions (insofar as pregnancy can be called a condition) are commonplace in medicine, and are dealt with through formal or informal collaboration such as that described between the Obstetrics and Gynaecology Directorate and the Endocrinology Directorate: to insist that any activity is a component of at most one other means that this situation is very difficult to describe (and so, by Occam's razor, the less flexible structure is 'wrong' in much the same way as the Ptolemaic description of the solar system is 'wrong' when compared to Kepler's).

As a result of these and other counter-examples, the representation of *During* was changed from a tree to a graph:

T3: $During: Activities \leftrightarrow Activities$.

During and *After* are given formal semantics in the theory through the description of their interactions with other state components. At this stage we have only met the partitions of *Activities*, *Before* and *After*. The relevant theorems concerning these sets and relations are:

I8: $During^{+} \cap id[Activities] = \varnothing$

I9: $(im\ Includes)\ Complete \subseteq Complete$

I10: $(im\ During)\ Proceed \subseteq Proceed$

I11: $(After \cup Before) \rhd Dom(During) \subseteq Includes \circ During$

During (and hence *Includes*) is a 'Directed Acyclic Graph': no component (or sub-component, or sub-sub-component, and so on) of an activity can be itself. Any activity that is a component of a completed activity must itself be a member of *Complete*, and similarly, if an activity is proceeding, then the activity of which it is a part (if any) is itself a member of *Proceed*. Thus a doctor cannot be still seeing a patient if that patient has left the clinic, and if the doctor is seeing a patient, then the clinic of which that doctor consultation is a part must have commenced (and not yet be complete).

The last invariant states that any two activities that are in the *After* relation that are each Included in some activity, must have at least one of those 'higher level' composite activities in common. This is part of the definition of *After* and *During*: if any two activities are ordered, it must be because they are part of a higher level activity that requires such an ordering.

9.3.5 The Graph as a Family: a Useful Metaphor.

The clumsiness of the explanation of the last invariant illustrates an important point. A very succinct piece of set theoretic notation might be very difficult to explain informally. This must be tried however, both as a courtesy to the reader and to aid interpretation of the concepts involved. Sometimes such informal explanations will inevitably be either clumsy or longwinded as complex ideas, expressed precisely as symbolic predicates, are translated into natural language. At other times however we can use the power of natural language to help us in our task. A powerful reasoning aid is the use of metaphor, and in many cases we can use a metaphor for the structure or theorem defined in the theory to help in its

informal exposition. An example of this that will be described here is the use of the family metaphor to help us understand the relation between elements of graphs.

A family tree can be understood as being a particular sort of graph. The main link in the classical family tree is from father to child. We are not interested in implicit sexual discrimination so we can say that the link we care about is from parent to child. A family tree of this type is given below, based around the graph is_parent_of, using the same style for representing graphs as was introduced earlier.

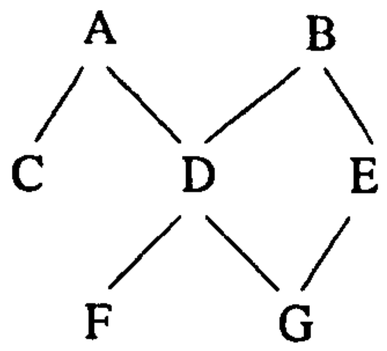


Figure 3-8: A representation of the Directed Acyclic Graph is_parent_of

In this case this can be represented as: {(A, C), (A, D), (B, D), (B, E), (D, F), (D, G), (E, G)}.

If we consider this as a family tree, then we can use conventional descriptions of family relations to describe selected pairs of elements. Thus A is a parent of C, and a parent of D. Someone's parent's child is a sibling, so C is a sibling of D. The relation is_sibling_of could be represented by the set: {(C, D), (D, C), (D, E), (E, D), (F, G), (G, F)}. It is usually considered to be the case that someone's child's parent is either that person or their spouse. Thus A is the spouse of B, and the relation is_spouse_of could be represented by the set: {(A, B), (B, A), (D, E), (E, D)}. Someone's ancestors consists of his or her parents, their parents, their parents and so on. Thus the ancestors of F are {D, A, B}. Descendants is the inverse of ancestors. Pushing the analogy a bit far, we might also say that A and B are orphans as neither has parents.

If we change the names of the sets, relations and values in the family tree, then we can represent any graph, and using the family as a metaphor, we can talk about the relation between two elements of this arbitrary graph as siblings, ancestors, spouses and so on. Thus if activity $a1$ *Includes* activities $a2$ and $a3$, we can say that $a2$ and $a3$ are siblings.

Using this metaphor to talk about the *Includes* graph, we can more easily express the last invariant by insisting that if one activity is after another, those two activities must be siblings.

9.3.6 Operations on *ActClass3*

The operations that are invoked in the class where *Includes* and *During* are introduced - ActClass3 - are different from those described in earlier classes, but taking advantage of the compositional calculus of the notation enables us to describe new operations as refinements of old ones. For example, we saw that an activity cannot be completed unless all its component activities have been finished also (Invariant I9). Thus the operation ***ActClass3.Complete***(a) is described in the notation as follows:

ActClass3.Complete(a)

$(im\ Includes)\ \{a\} \subseteq Complete$

ActClass2.Complete(a)

.

The precondition of this operation says that not only must the preconditions of ***ActClass2.Complete*** be satisfied (and any operations that inherits), but also the predicate

$(im\ Includes)\ \{a\} \subseteq Complete$

which says that all child activities of the activity to be completed - a - must be members of the set *Complete*.

Each operation that appeared in ActClass1 has been inherited by ActClass3, and appropriate preconditions specified. For example, when an activity is suspended, we must sure that it has no proceeding children. We do this through use of the precondition

$Includes\ \{a\} \subseteq Request \cup Complete.$

When we cancel an activity, we must be sure that all its descendants are members of the set *Request*:

$(im\ Includes^{+})\ \{a\} \subseteq Request.$

Not only that, but we must also ensure that those requests are themselves removed from the set *Activities* (if an activity is cancelled, then so are all its component parts). We do this through the use of a postcondition:

$(im\ Includes^{+})\ \{a\} \cap Activities' = \varnothing.$

ActClass2.Request(A_b,a_n) has been inherited and given the new name ***ActClass3.Create***(A_b,a_n) which creates an orphan activity and places it *After* all activities in the set A_b. The only new operation is ***ActClass3.Embed***(A_p,A_b,a_c). This creates a new activity a_c (c for 'child') and also places it *After* all activities in the set A_b (b for before): it also makes it a child of all activities in the set A_p (p for parent).

The preconditions define the states of the system where we want the operation to be valid. Remember that the notation dictates behaviour through the insistence that at all times the invariant is satisfied, and that additionally, if an operation is to be invoked, the precondition must be satisfied. After the operation has been invoked the invariant is still satisfied, as is the postcondition. As long as there is a possible state that contravenes neither the postcondition or the invariant, then (subject to the precondition), the operation can be invoked. The precise change to the model state that the operation makes is derived from formal arguments based around the notion of 'minimal change' (discussed briefly in Section 6.2.9). As we have seen, formal proof was little used in the exploration of the theory. For this reason, the pre-conditions were made sufficiently strong to ensure that the 'minimum change' was precisely what was required, and the changes to model states are easily deducible from the pre and post-conditions, and the class invariants.

The case of the ***.Cancel*** operation is an example where we needed to add additional strength to the postcondition. Without a postcondition, the operation would be formally valid, but not have the desired effect. The invocation of the ***.Cancel*** operation on the parent activity would remove it from the set *Activities*. After the operation, the semantics of the notation says that the state of a model of the theory is that which satisfies the postcondition and invariants, and is changed minimally compared to the state before the operation. Without the postcondition specified, one valid state of the model after invocation of the operation would have all the original children of the parent activity remaining as elements of *Request*, but all pairs in the *Includes* and *During* relations which have the parent activity as a member removed. As this state is closer to the original than the desired one where all 'child' activities are cancelled as well, it is

the one implied by the operation according to underlying semantics of the notation. Because of this we need to strengthen the postcondition accordingly.

Another example, where additional strength was added to the precondition of an operation is as follows. Consider the operation ***ActClass3.Embed***(A_p,A_b,a_c): in the schema for this operation, we insist that none of the potential parent activities of a_c - that is, the set A_p - are yet complete, and that all activities in A_b have a parent in A_p. These type declarations and preconditions are thus (in addition to those specified in ***ActClass3.InRequest***(A_b,a_n) which is the inherited operation):

A_p: *Activities\Complete*

$A_b \subseteq Dom(During \rhd A_p)$.

The postcondition states that (in addition to the postconditions of the inherited operation) a_c is the child of all activities in A_p, as defined by the new state component *During'*:

$\{a_c\} \times A_p \subseteq During'$.

Thus the final operation schema is:

ActClass3.Embed(A_p,A_b,a_c)

A_p: *Activities\Complete*
$A_p \cap In \neq \varnothing$
$A_b \subseteq Dom(During \rhd A_p)$
ActClass2.InRequest(A_b,a_n)
$\{a_c\} \times A_p \subseteq During'$

(The extra pre-condition enforces a property relating to the boundary of the domain). Now this postcondition could be satisfied without having to use the preconditions described. For example, we might not insist that no parent activities were members of *Complete*. This operation schema would look like this:

BadActClass3.Embed(A_p,A_b,a_c)

A_p: *Activities*
$A_p \cap In \neq \varnothing$
$A_b \subseteq Dom(During \rhd A_p)$
ActClass2.InRequest(A_b,a_n)
$\{a_c\} \times A_p \subseteq During'$

.

The effect of this operation on a model where some of A_p were members of *Complete* would be to take them from that set and insert them into either *Request* or *Proceed* in order that invariant I9 was not contravened. As we saw in the discussion of the activity state transition diagram, we want to be sure that

when an activity is complete, it stays complete. Thus the operation ***BadActClass3.Embed***(A_p,A_b,a_c) is not formally invalid, but displays properties that would be undesirable in a model, and are not observed in the domain. At all stages in the development of the theory care was taken to ensure that the class schemas were not only consistent in a formal sense, but also ensured that models of the theory displayed desirable properties. This is one interpretation of the directive of the scientific method which encourages us to embolden the theory and render it more falsifiable.

There are parallels with most of the above operations that apply only to external activities: these are ***Act3.OutCreate***(a), ***Act3.OutEmbed***(A_p,a_c), ***Act3.OutProceed***(a) and ***Act3.OutComplete***(a).

One final point should be made in this sub-section. If we inspect the operation *.Start*(a) in this class, we find the postcondition

$(im\ During^+)\ \{a\} \subseteq Proceed'$.

However, the invariant

$(im\ During)\ Proceed \subseteq Proceed$

from the state schema renders this unnecessary. We could rewrite this invariant as:

$\forall p: Proceed \bullet (im\ During)\ \{p\} \subseteq Proceed$.

As we know that following the operation a is an element of $Proceed'$, we can thus say

$(im\ During)\ \{a\} \subseteq Proceed'$.

Now if the parents of a are in *Proceed*, then so must be its 'grandparents': if the 'grandparents' of a are in *Proceed*, then so must be its 'great-grandparents', and so on. ie:

$(im\ During^+)\ \{a\} \subseteq Proceed'$

which is the postcondition. In other words the postcondition is derivable anyway from the invariant making its presence superfluous.

The use of a specification is as an aid to communication, not an elegant mathematical statement. Thus tautologies should be encouraged where their introduction acts as a reminder to the reader of a property that is logically implied but might be overlooked. In this way the postcondition of the *.Start* operation in this class acts as an *'aide memoire'* and, it is hoped, adds clarity to the presentation.

9.3.7 Conclusion

In this section we have investigated the nature and representation of two different graphs (and their inverses): *Includes* (and *During*) and *After* (and *Before*). The original interpretation of *After* was as a resource dependant ordering of *Activities*, but this was refuted by the example of the blood test. The new interpretation of After is as a medically meaningful ordering which thus always applies to the same patient. *During*, the inverse of *Includes* was introduced as a tree: an *activity* might be *During* no more than one other. This assertion is refuted through the examples of pregnancy care for diabetic patients, and shared care (between hospital and GP): the new theory thus defines *During* to be a graph with the attendant invariants re-worked.

9.4 *Types*

9.4.1 Introduction

We have still not said anything of great interest about medicine as we do not know what sort of activities we are dealing with. This lack is partly remedied through the definition of the medical type. This next section introduces the set *Types*, and explains how the class which introduces *Types* (TypeClass1) and the latest operational class featuring only *Activities* and its graphs (ActClass1) are brought together to form a composite class (ATClass1) which is defined in some detail. A behavioural model of the theory thus far presented is expressed, introducing the concept of animation.

9.4.2 Types

A type is the medical description of an activity, and as such it is an attribute, or descriptive term, of the activity. Some examples of types from the DEDC are Dr Telephone, DSN Consultation, DSN Education Session, Established Diabetes Mellitus Pregnancy Care, Finger Prick Test, First Dr Consultation and Followup Dr Consultation (and many others: the most recent instantiation to the Endocrine Directorate involves forty types). Other clinical domains have other types: for example types of activity relevant to the Dermatology Directorate might be General Dermatology Clinic, PUVA Visit (PUVA is a form of sun-lamp treatment), Dressing Session, and Urticaria Consultation. A good understanding of the activity types was a central theme of the analysis, as it is mainly through the type of an activity that a clinician (or anyone else) describes what he or she does. A diabetologist, when describing the clinics she runs will talk about the 'initial visit' and 'followup visit' rather than individual activities of those types. The set of types that is pertinent to the Endocrinology Directorate is given in Appendix 3.

The set of *Types* was initially chosen from a single new set:

Types: *Set*[T].

As explained in section 9.2.9: Limiting the Scope: Introducing Boundaries, it became necessary to distinguish between those activities that take place within the organisational boundary in which we are interested, and those that take place without. As we saw in the case of diabetic pregnancies, activities that take place outside the scope of the particular medical area we are currently modelling might still be of interest to us. To distinguish between the types of 'internal' activities and 'external' activities, the carrier set of *Types* was changed into a Cartesian product of two sets: the 'descriptive term' (Blood Test, Doctor Consultation, Limb Amputation) and the 'organisation' (Diabetes and Endocrine Day Centre, Chemical Pathology, Surgery). Thus we have

T7: *Types*: *Set*[DT ⇿ Org].

We note the name of the organisation we are currently interested in when specialising the theory in the set Home, where

T8: *Home*: *Set*[Org].

Home is intended to be a singleton set when the theory has been specialised: it is a set at all rather than just a value to enable class initialisation. In the case of a specialisation to the day centre Home would be {Endocrine Directorate}. We can then define the set *HomeTypes* as all types that are applicable to internal activities as follows:

T7: *HomeTypes*: DT ⇸ Org

and

I18: *HomeTypes* = *Types* ▷ *Home*.

As was explained in section 8.3.5: Structure and Value - Different Levels of Refutation, the behaviour specialisation state components, of which *Types* is an example, was not explored. There are rules determining what is or is not acceptable (such as that defining *HomeTypes*), but no investigation of how the 'rules of the game' might be changed as the game is being 'played'. In short what this means is that there are invariants over types (and other specialisation state components) and one operation called .***Specialise*** which sets up an initial value for a model. This can only be invoked when the set of activities and types is the empty set. This is a totally unrealistic operation that doesn't reflect any in the domain - it is included in the theory for the sake of completeness, and to enable models to be run.

9.4.3 The Class Structure so far

So far in the presentation of the theory, four classes have been described. The first was ActClass1 which introduces *Activities* and its partitions. The second was ActClass2 which introduces the graphs *Before* and After. The third class, ActClass3 describes two more graphs over the set *Activities*: *During* and *Includes*. Both of the last two of these three classes adds to its predecessor through a process known as inheritance (by the 'Object-Oriented' community) or refinement (by the 'Formal Methods' community). Thus a model of ActClass3 would incorporate *Activities*, its partitions, and the graphs *Before* and After as well as those relations introduced in the class.

The last class described, TypeClass1, did not need to refer to activities at all. The class introduces the set *Types*, the singleton set *Home*, and some derived state components; it also describes a (very primitive) operation of the class. In describing the way members of the set *Types* behave, and possible models of the class TypeClass1, there is no need to consider the set *Activities* at all. By considering how the various activity classes behave independently from the type class, and *vice versa*, we have achieved a 'separation of concerns', one of the keys to successful systems analysis [Cohen84].

Although it is useful to examine *Activities* and *Types* independently of each other, it is the interaction of the concepts described that helps us to understand the operational behaviour of the domain. Thus we want to say that all activities have types, and the types they have constrain the behaviour of the model in which we find them: to do this, we must compose the classes ActClass3 and TypeClass1 together. In the theory this class is called ATClass1 (Act Type Class1). The invariants that hold over the composite class constrain the interactions of the state components that have been defined in the component classes. This is achieved partly through the use of relations between state components from each of the component classes, partly through predicates involving these state components and derived relations, and partly through the introduction of state components derived solely from one class, but only having significance as regards their interaction with the other class.

The facility with which we can investigate aspects of the domain in isolation, and then examine the interaction between these aspects is aided by the structure of the notation that was used in the work, specifically the schema calculus which is in many ways more powerful than those of the more common model based specification notations Z and VDM.

The class structure as it has been described so far is illustrated by the following diagram.

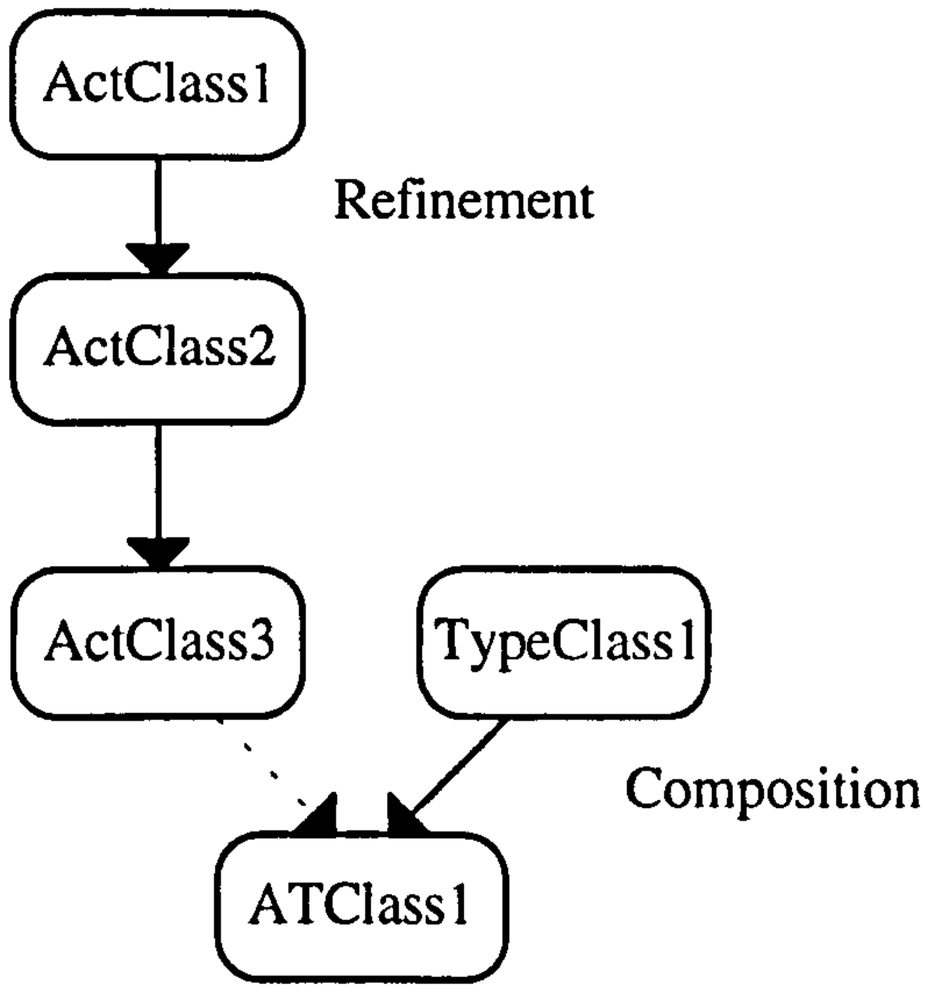


Figure 3-9: An Illustration of the class structure described so far.

ActClass2 is a refinement of ActClass1. ATClass1 is a composition of ActClass3 and TypeClass1 (See the conclusion of this chapter for an explanation of the dotted line between ActClass3 and ATClass1)

The theory as it was developed is not structured exactly like this - patients are introduced before activity types. It was considered that the introduction of types at this stage was more helpful in terms of the explanation of the theory.

9.4.5 Interactions Between *Types* and *Activities*: The Class ATClass1

We are interested in the concept of types only inasmuch as it can be applied to activities. This we do with the total function *ActType*:

T9: $ActType: Activities \rightarrow Types;$

all activities have exactly one type. Each structure which holds over types has some role to play in the constraining of possible model behaviours, or more specifically over possible activity structures. Thus with the quantity *HomeTypes* which was defined in the last section, we have two invariants:

I21: $(im\ ActType)\ In \subseteq HomeTypes$

I22: $(im\ ActType)\ Out \cap HomeTypes = \emptyset.$

Or, informally, there is no activity in the set *In* that has a type that is not in *HomeTypes*, and there is no activity in the set *Out* that has a type that is in *HomeTypes*. Another invariant that we introduce at this stage is one that constrains possible models of *Includes*:

I23: $ActType \circ Includes^{+} \circ ActType^{-1} \cap id[Types] = \emptyset$

which says that no activity can have a descendant that is the same type as it is (this invariant is similar in form to the one discussed later that insists that *Can_include* is a directed Acyclic graph).

The last two invariants that are pertinent to this class are:

I26: $\#ActType \circ (Includes \rhd Proceed) = \#(Includes \rhd Proceed)$,

I27: $\#ActType \circ (Includes \rhd Request) = \#(Includes \rhd Request)$.

These two predicates are similar in structure: the first says that for any activity, only one component activity of a given type can be a member of *Proceed*, the second that only one component activity of a given type can be a member of *Request*. Thus a doctor can only order one blood test at a time (though he can order one at the same time as another is being carried out), and a specialist nurse can book only one followup appointment as a result of a patient visit. These invariants were tentatively suggested as a means of tidying up the structures of possible models, but they have not yet been refuted and are useful later on in the theory when we want to be sure of which activity we are talking (ie we can say the request that is included in activity *a*1 rather than a request).

When creating new activities, the types of those activities must be given: thus ***ATClass3.Create***, ***ATClass3.Embed*** and ***ATClass3.SuddenStart*** (and their equivalents that apply to external activities) have an extra argument each, specifying the type of a_n and a_c respectively. In addition these operations have preconditions ensuring that invariants I26 and I27 are not contravened. The operations that move activities from one partition of *Activities* to another do not change the type of the activity, so they have no new arguments.

9.4.6 A Model of the Theory So Far

It is all very well talking about the theory in abstract, and examining some possible models that do not contravene any invariants, but we have not yet got any feel for how the domain that we are theorising about behaves. This we do by constructing behavioural models, or animations. The way we do this is by creating an initial (specialised) model and observing how its state components change when we invoke operations on it. We can show formally at each stage that the model does not contravene the invariants. To do so would lead to vast numbers of proofs of fairly trivial results (this level of theorem proving or model validation is best done by computer, and various animation programs exist for a number of formal notations - for example OBJ3 and 2OBJ. An animator does not yet exist for the Schuman-Pitt notation, but one - SUZAN - is currently being developed at The University of Surrey). Instead, the formal invariants are compared with the formal model informally. Some of this informal reasoning is given below.

The state of the domain model is given by values of the sets *Activities*, *Request*, *Proceed*, *Complete*, *Before*, *During* and *ActType*. The initial state (after invoking the specialisation operation) is recorded by 'initialising' all these sets to the empty set:

$Activities = Request = Proceed = Complete = After = Includes = ActType = \varnothing$.

The operations that have been defined in the five classes described are: ***.Create***, ***.Embed***, ***.SuddenStart***, ***.Start***, ***.Suspend***, ***.Complete***, ***.Cancel***, ***.OutCreate***, ***.OutEmbed***, ***.OutProceed***, ***.OutComplete***, and ***.Specialise***. Confining ourselves to internal activities, and ignoring the ***.Specialise*** operation that has already been invoked, and cannot be invoked again, we have seven operations to choose from (in addition, we shall not use ***.SuddenStart*** at the moment).

The only operation we can invoke from our initial 'empty' state is ***ATClass1'.Create***(A_b,t_n→a_n)[xiv] - all the others have preconditions that require the existence of elements of the set *Activities*. We must invoke this operation and supply the required parameters. Suppose we create a new activity of type Doctor Care:

ATClass1'.Create(∅,*Doctor Care*→*a*1)

Note that it is the model itself that 'provides' the activity name, chosen at random from the set A\/*ctivities*. We assume here that it has provided one called *a*1. The new state of the model is:

Activities = {*a*1}
Request = {*a*1}
Proceed = ∅
Complete = ∅
After = ∅
Includes = ∅
ActType = {(*a*1, *Doctor Care*)}

Using the graphical notation introduced in Section 8.3.3 above, we can represent the current state of the model as:

a1, Doctor Care

Doctor care is comprises a number of sub activities such as Initial Doctor Consultation (Init Dr Cons), Followup Doctor Consultation (Followup Dr Cons) and Blood Test. The allowable hierarchies have not been described by the classes described so far, so *a*1 could equally well include activities of type Diabetes Care, or Specialist Nurse Consultation. It is not the intention of the author to mislead, so we will only consider reasonable structures (neither prescribed nor prohibited by the theory so far) for the model.

We next invoke the operation

ATClass1'.Embed({*a*1},∅,*Init Dr Cons*→*a*2)

whereupon the state of the model is

a1, Doctor Care
|
a2, Init Dr Cons

Activities = {*a*1, *a*2}
Request = {*a*1, *a*2}
Proceed = ∅
Complete = ∅
After = ∅
Includes = {(*a*1, *a*2)}
ActType = {(*a*1, *Doctor Care*), (*a*2, *Init Dr Cons*)}

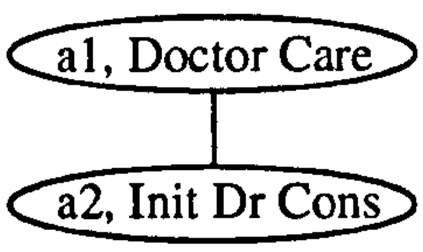
From now on the state of an animation will be given in two columns - the right hand column expressing the state of the model as values of sets, the left hand column being a graphical representation of the same thing.

We cannot have *a*2 a member of *Proceed* unless *a*1 is also (invariant I10). However, as we have seen ***.Start*** can be invoked using a child of a request as its argument - the operation has a postcondition which ensures that all parents are members of *Proceed*'. The operation we invoke in the model is:

AT1.Start(*a*2),

[xiv] This operation is marked with the prime symbol as the equivalent in the actual theory has an additional argument in the form of a patient identifier. Patients have not yet been introduced, so the argument has been left out here.

after which the state of the model is:



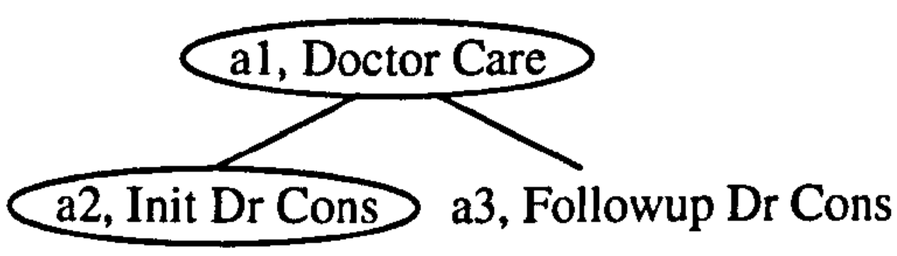
Activities = {*a*1, *a*2}
Request = ∅
Proceed = {*a*1, *a*2}
Complete = ∅
After = ∅
Includes = {(*a*1, *a*2)}
ActType = {(*a*1, *Doctor Care*), (*a*2, *Init Dr Cons*)}

This diagram introduces a new graphical representation: any activity in a 'bubble' is in the set *Proceed*, and any activity in a 'box' is in the set *Complete* (any activity without a border is in *Request*).

To represent the doctor ordering a followup visit, we invoke the operation

ATClass*1'.*Embed({*a*1},{*a*2},*Followup Dr Cons*→*a*3)

to give us



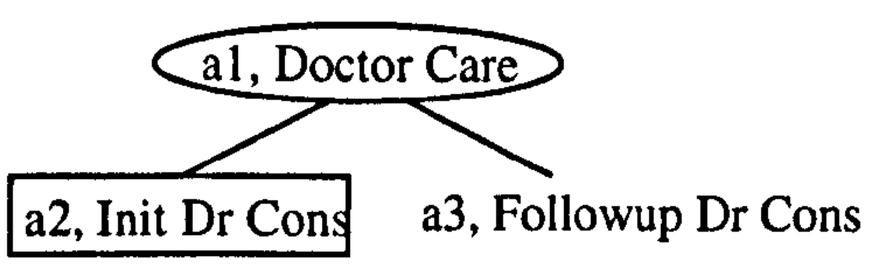
Activities = {*a*1, *a*2, *a*3}
Request = {*a*3}
Proceed = {*a*1, *a*2}
Complete = ∅
After = {(*a*3, *a*2)}
Includes = {(*a*1, *a*2), (*a*1, *a*3)}
ActType = {(*a*1, *Doctor Care*), (*a*2, *Init Dr Cons*), (*a*3, *Followup Dr Cons*)}

Here the activity *a*2 has been placed in the relation *Before* with the new activity (not shown on the graphical representation).

Now because *a*3 is After *a*2, *a*3 cannot be started until *a*3 has finished. In other words, the operation ***ATClass*1.*Start***(*a*3) is forbidden in this state by its precondition. We must first complete activity *a*2 by invoking the operation

ATClass*1.*Complete(*a*2)

to give us the state

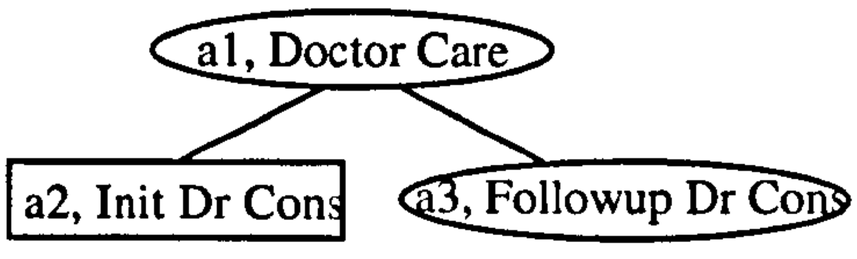


Activities = {*a*1, *a*2, *a*3}
Request = {*a*3}
Proceed = {*a*1}
Complete = {*a*2}
After = {(*a*3, *a*2)}
Includes = {(*a*1, *a*2), (*a*1, *a*3)}
ActType = {(*a*1, *Doctor Care*), (*a*2, *Init Dr Cons*), (*a*3, *Followup Dr Cons*)}.

Now we can start the followup activity using the operation

ATClass*1.*Start(*a*3)

which gives



Activities = {*a*1, *a*2, *a*3}
Request = ∅
Proceed = {*a*1, *a*3}
Complete = {*a*2}
After = {(*a*3, *a*2)}
Includes = {(*a*1, *a*2), (*a*1, *a*3)}
ActType = {(*a*1, *Doctor Care*), (*a*2, *Init Dr Cons*), (*a*3, *Followup Dr Cons*)}

The behaviour seen in this model is not very informative, but it does give a feel for how possible behaviours of the model are constrained by the theory. As the theory gets richer these behavioural models, or animations, become more useful.

9.4.7 Conclusion

This section has discussed the (medical) *Types* which is a specialisation state component, and its interaction with *Activities*, an operational state component. *Types* is given values via an operation called *.Specialise* that can only be invoked once, immediately after the model has been initialised. Some values of *Types* specific to our main exemplar medical domain, the DEDC, are given above. A model of the theory can be expressed using set extension and an appropriate graphical representation. A behavioural model of the theory can be explored by animating a static model. Operations are invoked with real values as their arguments, and the state of the model can be examined before and after the invocation.

9.5 Structures over *Types*

9.5.1 Introduction

Now that we have introduced types and ensured that every activity has one we can start to impose constraints on activity structures that are specific to the particular medical domain we are looking at. The two classes we are interested in here are TypeClass2 and ATClass2. TypeClass2 is a refinement of TypeClass1, and introduces a structure over the set *Types*. The structure was initially a graph but was re-defined as a more complex concept. Most of the ideas behind the structure introduced in this class can be understood by considering the earlier and simpler relation, so we will investigate this first. TypeClass2 is composed with ATClass1 to give ATClass2 which describes the interaction between the specialisation state component *Can_include* (or its subsequent derivative) and the operational state components relating to *Activities*. An animation of a model of this class would be constrained by the particular structure created on specialising the theory to the medical area of interest, and so could only be a model of that area (with the previous operational class, ATClass1, an animation could be made to be a model of any medical area just by changing the names of the types - the same cannot be done once activity constraining structure over *Types* is introduced). For this reason, it is necessary to use the correct values of *Types* and its structures when specialising the theory - discovering what these values were is an important part of the analysis.

9.5.2 The Graph *Can_include*

The first structure introduced was the graph *Can_include*, defined as follows:

Can_include: *Types* ⇿ *Types*.

The role of this relation is to act as a constraint over possible *Includes* hierarchies - an activity of type $t1$ can be *During* an activity of type $t2$ only if $t1$ *Can_include* $t2$. The only invariants pertinent to this (hypothetical) class are

$$Can\_include^{+} \cap id[Types] = \varnothing$$

and

$$Can\_include \cap (Types \backslash HomeTypes)^2 = \varnothing$$

(These are not given invariant numbers because they do not appear in the final theory, their function here being to give the relation *Can_include* some formal semantics). The first invariant says that *Can_include* is an acyclic graph, the second that if two types are in the *Can_include* relation, then one of those types must be in *HomeTypes*.

In defining *Can_include* to be a directed acyclic graph, we are limiting quite severely the sorts of things we can talk about: in particular we cannot include any generic types as well as their subtypes in the set Types. An example of a generic type might be 'test'. An activity might well be a test, so why should we not say that the type is permissible? The problem comes if we consider subtypes of the test. For example a blood analysis is also a test, and a complex test might well have a blood analysis as one of its components. We could say that test *Can_include* blood analysis, but as the analysis is also a test, we would have to say that test *Can_include* test which we have disallowed through the invariant. We must be careful how we choose the types with which to populate the specialisation state components: we must ensure that we never have two members of *Types*, $t1$ and $t2$, such that we can say a $t1$ is a $t2$ or *vice versa*.

The second invariant reflects on the nature of the boundary we set at any specialisation. The boundary has a degree of fuzziness about it in that an internal activity can include external activities (The Home organisation might 'sub-contract' some of its services to other organisations - blood tests carried out for the Endocrinology directorate by the Clinical Chemistry Directorate for example) and an external activity can include an internal one (where the Home organisation is itself acting as the sub-contractor as in a reversal of the situation above, or the situation where Diabetic Specialist Nurse support is given by the Diabetes and Endocrine Day Centre to a GP - a form of 'shared care'). However, we never need to know how the external activities choose to structure themselves internally to their own organisations. For example a blood test, carried out by the Clinical Chemistry directorate might be composed of several subsidiary activities, involving different health care professionals. Unless the Clinical Chemistry directorate is our Home organisation we do not need to know this, so this 'internal' structure would not be recorded in a model of the theory. We can insist on this by saying that an external type can never be in the relation *Can_include* with another external type, as in the second invariant.

As with the TypeClass1, TypeClass2 is composed with a class defining activity behaviour. This is done with ATClass1 to give the composite class OldATClass2. There is only one invariant in this class:

$$ActType \circ Includes \circ ActType^{-1} \subseteq Can\_include$$

which says that for any pair of activities in *Includes*, the corresponding pair made of the types of those activities must be in *Can_include*. The operations where we must consider this invariant are ***.Embed***, its derivative ***.SuddenStart***, and its parallel for external activities ***.OutEmbed***. Thus a precondition of the operation

OldATClass2.Embed$(t_c,a_p,A_b \rightarrow a_c)$

is

$(ActType(a_p),t_c) \in Can\_include$

or the type of the parent activity *Can_include* the type of the new child activity.

We can see here why this early version of a structure over *Types* might was abandoned - it was sufficient to constrain behaviour while *During* was a tree (as is supported by this operation where the child activity can only be embedded in one parent), but is too simple to be able to apply to the present theory where *During* is represented as a graph.

Before we examine a new structure over *Types* that supports this multiple parentage of activities, we can use the simpler version to investigate how a proposed specialisation of the theory was refuted, and a more realistic one was constructed in its place.

9.5.3 How Important is the Visit?

Once the theory was considered sufficiently robust, it was specialised to be relevant to an area of the hospital - the Diabetes Day Centre. This was done by giving values to the elements of the set *Types* and the relations over that set. Again, the rigour of set theory helped in this task by making predictions that could be refuted. Initially, the analyst and the 'tame clinician' who had agreed to help in the project viewed the process of care as delivered by the centre to be grouped by 'visit'. The first activity arranged for the patient was a visit to the day centre. This visit comprised of a Doctor Consultation, a Dietitian Consultation, a Specialist Nurse Consultation and a Blood Test. The Specialist Nurse Consultation might be followed up by others over several weeks. Thus the visit was considered an abstract event that took place over a prolonged period of time. The instantiation of the set types was then:

{*GP Care*, *Hospital Diabetic Care*, *First Visit*, *Followup Visit*, *Doctor Consultation*, *Blood Test*, *Dietetic Care*, *Specialist Nurse Care*, *Dietetic Consultation*, *Specialist Nurse Consultation*} $\subseteq$ *Types*

and the relation *Can_include* could be represented by the following graph (DSN is the abbreviation for Diabetic Specialist Nurse):

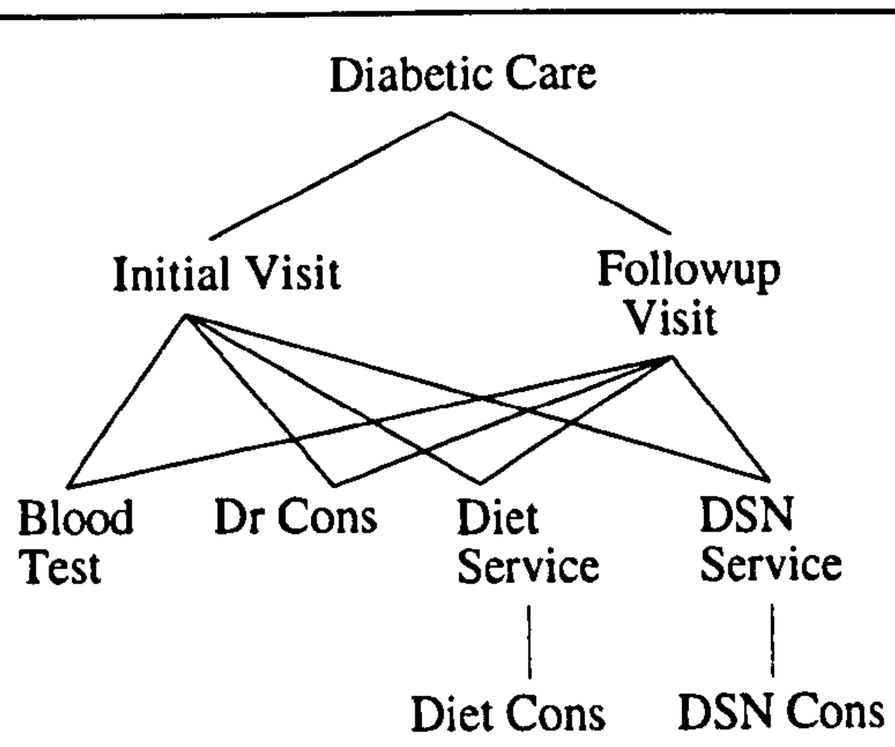


Figure 3-10: Graph of initial *Can_include* relation over the set *Types* named above.

Thus every dietetic consultation is associated with a dietetic care activity and a 'visit'. There is generally one doctor consultation per visit, the doctor creating a followup visit at the end of the consultation, so each dietetic consultation could be associated with a particular doctor consultation. This seemed to be the way things happened - the doctor deciding whether or not a patient needed to see the dietitian or specialist nurse at each visit, delegating authority to treat to the nurse or paramedic, and then regaining control in time for the next visit. One of the implications of this structure is that as one followup visit is generally After the preceding one, the dietitian and specialist nurse must have finished the care they are dispensing before the next doctor consultation. We can see this in the following behavioural model which represents a typical sequence of care for a new patient as perceived under the original specialisation. Note that this is a typical sequence of care - not the standard, or the necessary, or the recommended. There are many possible behavioural sequences, far too many for us to investigate (the number of possible behavioural traces will increase roughly exponentially with the number of operations invoked). All we can hope to do is see if there are real behaviours of the domain that cannot be modelled using the theory: the identification of these requires knowledge of both the theory and the domain. The following is an example of such a behavioural refutation.

Initially, all sets are empty. We first invoke the ***.Specialise*** operation to specialise the theory to the Diabetes and Endocrine Day Centre. Consequent to this operation, the specialisation state components have values as described in the graph above, and the operational state components will remain unchanged, all being equal to the empty set:

$Activities = \varnothing$
$Request = \varnothing$
$Proceed = \varnothing$
$Complete = \varnothing$
$After = \varnothing$
$Includes = \varnothing$
$ActType = \varnothing$.

The first ('operational') operation we invoke is

OldATClass2.Create({},*Diabetic Care*→*a*1)

to represent the initial referral of the patient to the clinic. The result of the operation is:

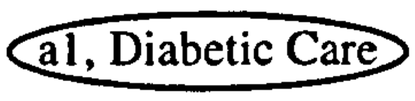
a1, Diabetic Care

$Activities = \{a1\}$
$Request = \{a1\}$
$Proceed = \varnothing$
$Complete = \varnothing$
$After = \varnothing$
$Includes = \varnothing$
$ActType = \{(a1, Diabetic\ Care)\}$

After the creation of this activity, which will span all the care provided to the patient by the day centre, generally an agreement to treat the patient will have to be made by the relevant consultant after which we can claim that the activity has started:

OldATClass2.Start(*a*1)

which gives

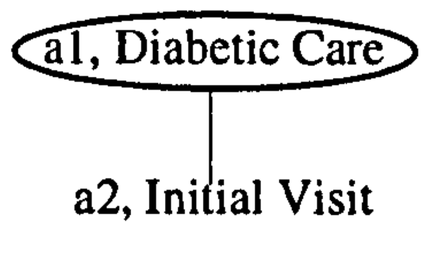


Activities = {*a*1}
Request = ∅
Proceed = {*a*1}
Complete = ∅
After = ∅
Includes = ∅
ActType = {(*a*1, *Diabetic Care*)}

As shown in the illustration of the *Can_include* relation, there are only two types of activity that can be embedded in Diabetic Care: Initial Visit and Followup Visit. Even if the patient is an established patient at another clinic, and has started attending the day centre at St Thomas' (maybe because the patient has moved house), they will (almost always) still attend the 'initial visit' first. So the usual operation invoked is:

OldATClass2.Embed(*a*1,∅,*Initial Visit*→*a*2)

to give

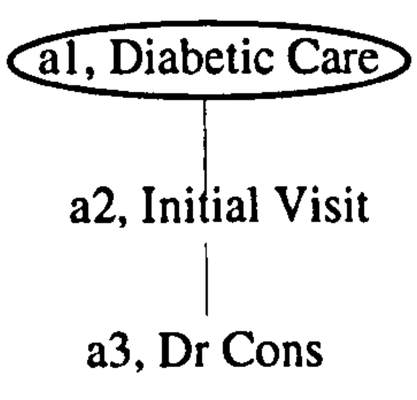


Activities = {*a*1, *a*2}
Request = {*a*2}
Proceed = {*a*1}
Complete = ∅
After = ∅
Includes = {(*a*1, *a*2)}
ActType = {(*a*1, *Diabetic Care*), (*a*2, *Initial Visit*)}

Note that the form of ***.Embed*** we use here is different from that described for ATClass1 in that a single parent activity is provided as an argument instead of a set.

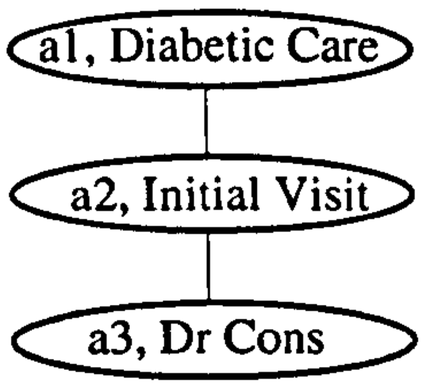
The initial visit typically consists of a blood test, a doctor consultation, and a visit to the specialist nurse or dietitian. The analysis seemed to reveal a longer sort of visit which included a sequence of specialist nurse and dietitian consultations. These consultations were provided as parts of the service that the paramedics were requested (by the doctor) to deliver to the patient. In short, the observed sequence of operations and subsequent model states was

OldATClass2.Embed(*a*2,∅,*Dr Cons*→*a*3)



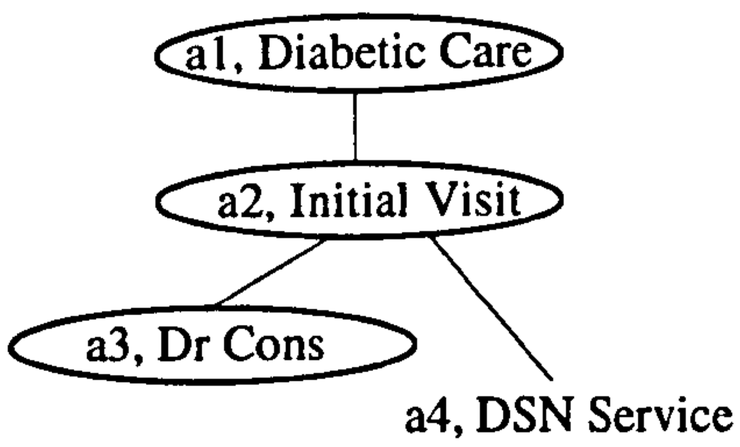
Activities = {*a*1, *a*2, *a*3}
Request = {*a*2, *a*3}
Proceed = {*a*1}
Complete = ∅
After = ∅
Includes = {(*a*1, *a*2), (*a*2, *a*3)}
ActType = {(*a*1, *Diabetic Care*), (*a*2, *Initial Visit*), (*a*3, *Dr Cons*)}

OldATClass2.Start(*a*3)



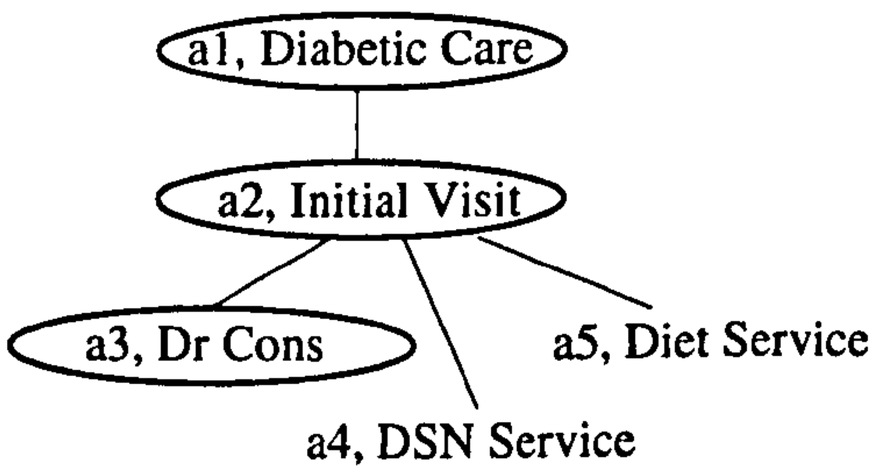
Activities = {*a*1, *a*2, *a*3}
Request = ∅
Proceed = {*a*1, *a*2, *a*3}
Complete = ∅
After = ∅
Includes = {(*a*1, *a*2), (*a*2, *a*3)}
ActType = {(*a*1, *Diabetic Care*), (*a*2, *Initial Visit*), (*a*3, *Dr Cons*)}

OldATClass2.Embed(*a2,∅,DSN Service→a4*)



Activities = {*a1*, *a2*, *a3*, *a4*}
Request = {*a4*}
Proceed = {*a1*, *a2*, *a3*}
Complete = ∅
After = ∅
Includes = {(*a1*, *a2*), (*a2*, *a3*), (*a2*, *a4*)}
ActType = {(*a1*, *Diabetic Care*), (*a2*, *Initial Visit*), (*a3*, *Dr Cons*), (*a4*, *DSN Service*)}

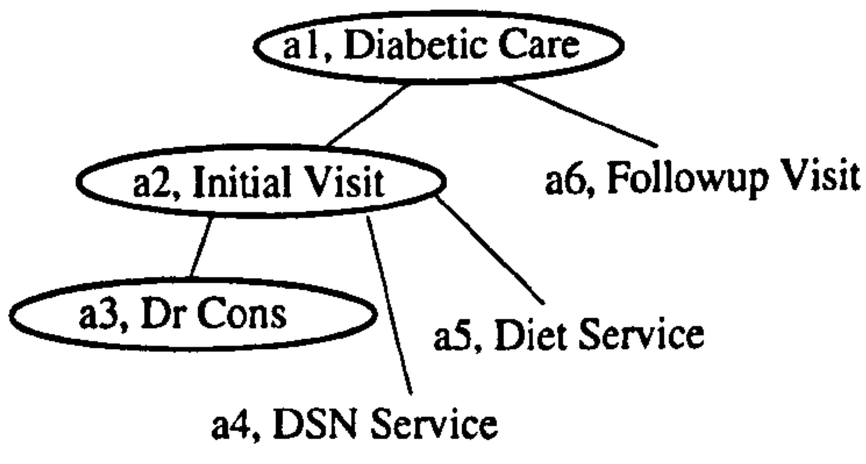
OldATClass2.Embed(*a2,∅,Dietitian Service→a5*)



Activities = {*a1*, *a2*, *a3*, *a4*, *a5*}
Request = {*a4*, *a5*}
Proceed = {*a1*, *a2*, *a3*}
Complete = ∅
After = ∅
Includes = {(*a1*, *a2*), (*a2*, *a3*), (*a2*, *a4*), (*a2*, *a5*)}
ActType = {(*a1*, *Diabetic Care*), (*a2*, *Initial Visit*), (*a3*, *Dr Cons*), (*a4*, *DSN Service*), (*a5*, *Diet Service*)}

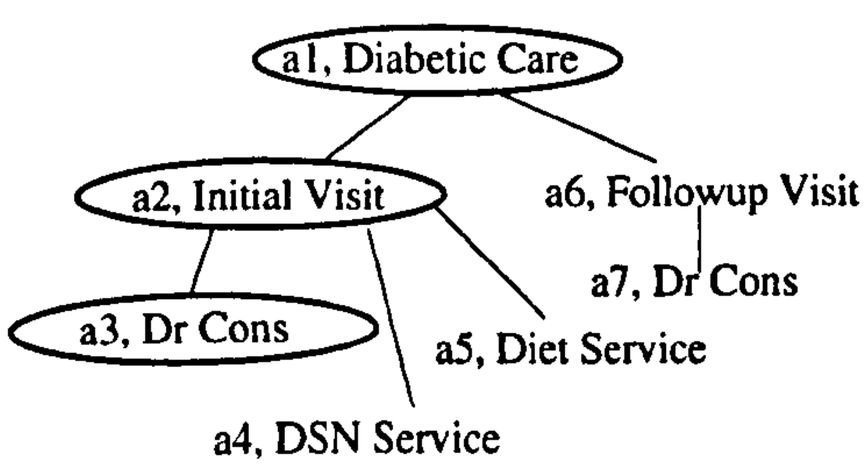
Before the doctor finishes the consultation, he or she will almost certainly arrange for the patient to come back for a followup appointment in six month's time or so. To do this, a followup visit must be created, and another Dr Cons embedded within it. The Followup Visit is *After* the Initial Visit, and this is reflected in the operations invoked which are as follows:

OldATClass2.Embed(*a1,{a2},Followup Visit→a6*)



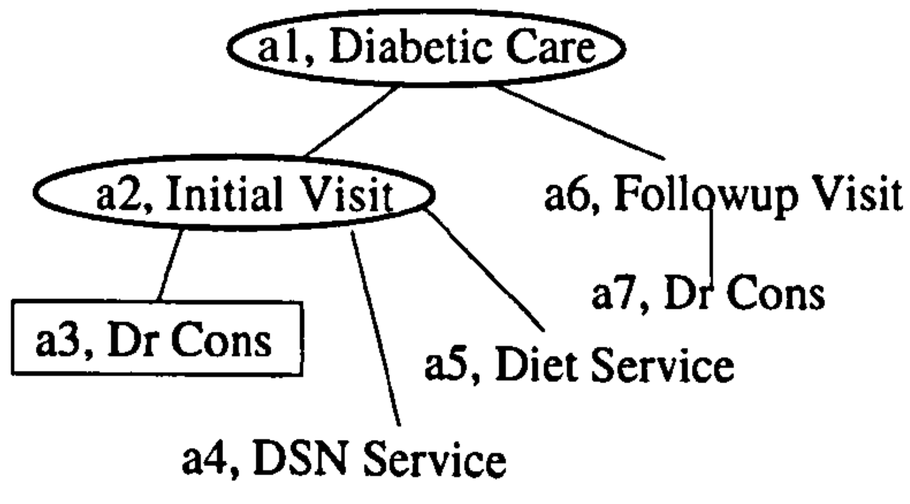
Activities = {*a1*, *a2*, *a3*, *a4*, *a5*, *a6*}
Request = {*a4*, *a5*, *a6*}
Proceed = {*a1*, *a2*, *a3*}
Complete = ∅
After = {(*a6*, *a2*)}
Includes = {(*a1*, *a2*), (*a2*, *a3*), (*a2*, *a4*), (*a2*, *a5*), (*a1*, *a6*)}
ActType = {(*a1*, *Diabetic Care*), (*a2*, *Initial Visit*), (*a3*, *Dr Cons*), (*a4*, *DSN Service*), (*a5*, *Diet Service*), (*a6*, *Followup Visit*)}

OldATClass2.Embed(*a6,∅,Dr Cons→a7*)



Activities = {*a1*, *a2*, *a3*, *a4*, *a5*, *a6*, *a7*}
Request = {*a4*, *a5*, *a6*, *a7*}
Proceed = {*a1*, *a2*, *a3*}
Complete = ∅
After = {(*a6*, *a2*)}
Includes = {(*a1*, *a2*), (*a2*, *a3*), (*a2*, *a4*), (*a2*, *a5*), (*a1*, *a6*), (*a6*, *a7*)}
ActType = {(*a1*, *Diabetic Care*), (*a2*, *Initial Visit*), (*a3*, *Dr Cons*), (*a4*, *DSN Service*), (*a5*, *Diet Service*), (*a6*, *Followup Visit*), (*a7*, *Dr Cons*)}

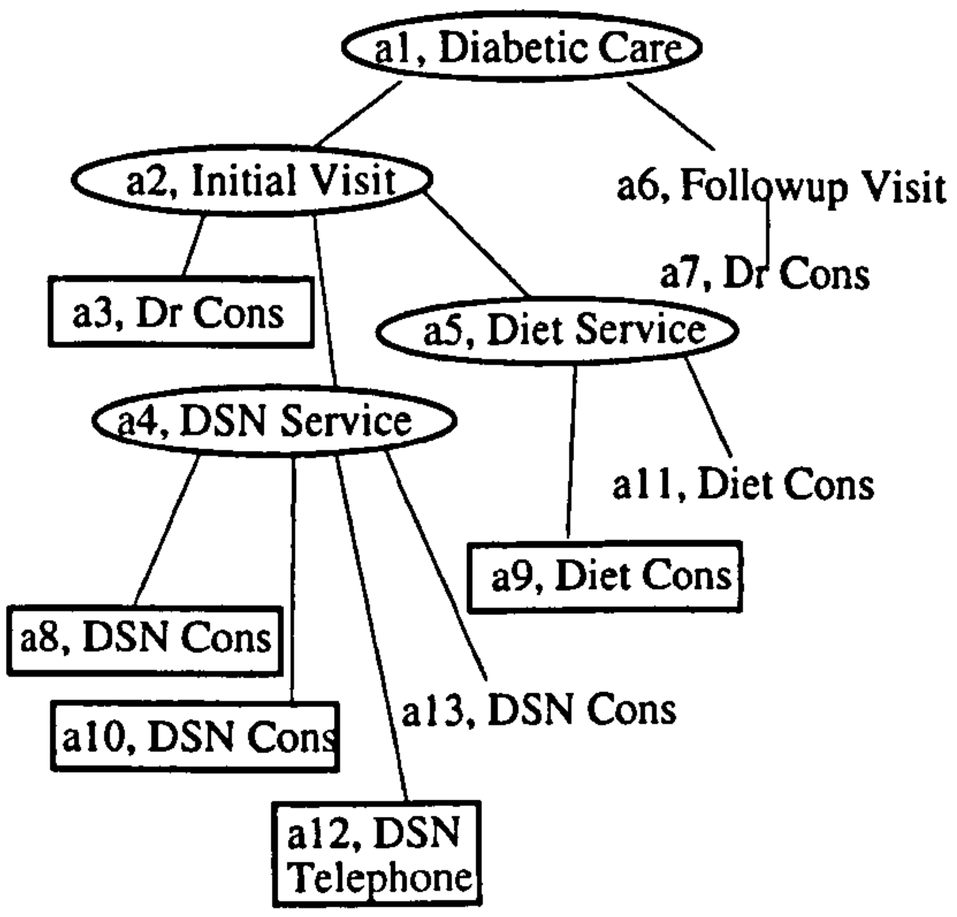
OldATClass2.Complete(*a*3)



Activities = {*a*1, *a*2, *a*3, *a*4, *a*5, *a*6, *a*7}
Request = {*a*4, *a*5, *a*6, *a*7}
Proceed = {*a*1, *a*2}
Complete = {*a*3}
After = {(*a*6, *a*2)}
Includes = {(*a*1, *a*2), (*a*2, *a*3), (*a*2, *a*4), (*a*2, *a*5), (*a*1, *a*6), (*a*6, *a*7)}
ActType = {(*a*1, *Diabetic Care*), (*a*2, *Initial Visit*), (*a*3, *Dr Cons*), (*a*4, *DSN Service*), (*a*5, *Diet Service*), (*a*6, *Followup Visit*), (*a*7, *Dr Cons*)}

The discharging of the DSN Service is the responsibility of the diabetic specialist nurse; the discharging of the Diet Service the responsibility of the dietitian. Both of these services will generally be discharged through the provision of a succession of consultations (specialist nurse or dietitian), blood tests, telephone consultations and so on. A typical sequence of these is given below along with the eventual state.

OldATClass2.Embed(*a*4,∅,*DSN Cons*→*a*8)
OldATClass2.Start(*a*8)
OldATClass2.Embed(*a*4,{*a*8},*DSN Cons*→*a*10)
OldATClass2.Complete(*a*8)
OldATClass2.Embed(*a*5,∅,*Diet Cons*→*a*9)
OldATClass2.Start(*a*9)
OldATClass2.Embed(*a*5,{*a*9},*Diet Cons*→*a*11)
OldATClass2.Complete(*a*9)
OldATClass2.Start(*a*10)
OldATClass2.Embed(*a*4,{*a*10},*DSN Cons*→*a*13)
OldATClass2.Complete(*a*10)
OldATClass2.SuddenStart(*a*4,*DSN Telephone*→*a*12)
OldATClass2.Complete(*a*12)



Activities = {*a*1, *a*2, *a*3, *a*4, *a*5, *a*6, *a*7, *a*8, *a*9, *a*10, *a*11, *a*12, *a*13}
Request = {*a*6, *a*7, *a*11, *a*13}
Proceed = {*a*1, *a*2, *a*4, *a*5}
Complete = {*a*3, *a*8, *a*9, *a*10, *a*12}
After = {(*a*6, *a*2), (*a*10, *a*8), (*a*13, *a*10), (*a*11, *a*9)}
Includes = {(*a*1, *a*2), (*a*2, *a*3), (*a*2, *a*4), (*a*2, *a*5), (*a*1, *a*6), (*a*6, *a*7), (*a*4, *a*8), (*a*5, *a*9), (*a*4, *a*10), (*a*5, *a*11), (*a*4, *a*12), (*a*4, *a*13)}
ActType = {(*a*1, *Diabetic Care*), (*a*2, *Initial Visit*), (*a*3, *Dr Cons*), (*a*4, *DSN Service*), (*a*5, *Diet Service*), (*a*6, *Followup Visit*), (*a*7, *Dr Cons*), (*a*8, *DSN Cons*), (*a*9, *Diet Cons*), (*a*10, *DSN Cons*), (*a*11, *Diet Cons*), (*a*12, *DSN Telephone*), (*a*13, *DSN Cons*)}

Inspection shows that at no stage are the state invariants contravened, nor any pre-condition violated. Two invariants to remember are I26 and I27 that prevent their being any more than one request (or, respectively, proceeding activity) of a given type included in a single activity. This means that only one paramedic followup visit can be booked at a time.

The important activities here are the paramedic consultations that have not yet been completed: *a*11 and *a*13. Because the Initial Visit (*a*2) is *Before* the Followup Visit (*a*6) which includes the followup Dr Cons (*a*7), the next visit to the day centre cannot take place until the initial visit is complete, which in turn

entails the completion of all the paramedic activities which are descendants of *a*2. This seemed reasonable enough as generally the DSN Service and Diet Service activities are complete within a matter of weeks, and the followup visit will still be several months away.

While presenting this model at a departmental meeting it was claimed (by one of the clinicians present) that this was a poor representation of how things actually worked in the clinic. The specialist nurse and dietitian might continue to dispense care irrespective of when the doctor sees the patient. The patient might refer themselves to the specialist nurse many months after the first visit, and be seen by him or her regularly over a period which spanned the first followup visit. This is not permissible in the specialisation described, and even if it were, we would not be able to represent the perception of the majority of the clinicians which was to accord nurses and other paramedics a significant degree of professional autonomy. A new specialisation was developed which takes this into account, and relegates the 'visit' concept to a scheduling detail rather than the central component of Diabetes Care. The new specialisation of the theory is illustrated in the following figure (of a subset of the new *Can_include*):

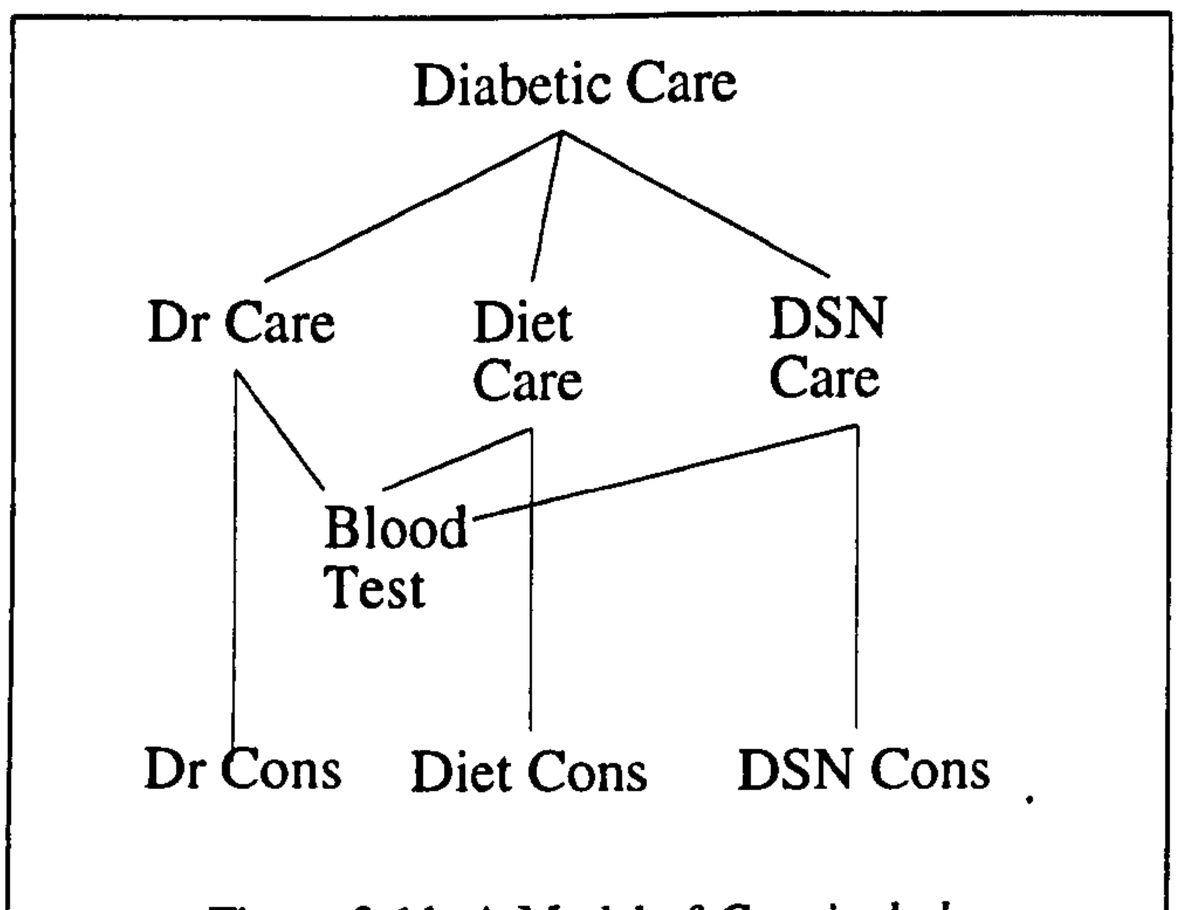


Figure 3-11: A Model of *Can_include*

The new specialisation can support the same order of activities as the old one, and can also represent those occasions when the specialist nurse or dietitian act independently of the doctor. It is important that models of both sorts are permitted, for to simply change from allowing one group to allowing another would imply that the observations that lead to the creation of the first specialisation were wrong. The observations, elicited from clinicians, were not wrong - it was the underlying structure, in the form of the values of the specialisation state components, that was wrong and that it was insufficiently flexible to support the circumstances that came to light in the departmental meeting.

9.5.4 Compulsory Activities: the Brief Appearance of *Comprises* and *Requires*.

Early versions of the theory had two additional structures over the set *Types*. The first was *Comprises* which was a subset of *Can_include*, the second *Requires* which was a triple over the set. In the old theory, the structures were declared and defined as:

Comprises: *Types* $\leftrightarrow\!\!\!\!\!+\!\!\!\!\!\rightarrow$ *Types*

Requires: *Types* $\rightarrow\!\!\!\!\!+$ (*Types* $\leftrightarrow\!\!\!\!\!+\!\!\!\!\!\rightarrow$ *Types*)

with the invariants

Comprises $\subseteq$ *Can_include*

and

$\forall t$: *Types* • *Dom*(*Requires*(t)) $\cup$ *Cod*(*Requires*(t)) $\subseteq$ *Im Comprises* {t}

The thinking behind these two quantities was as follows. *Activities* of certain types always seem to have a certain minimum structure in terms of component activities. Thus (it was thought) an initial visit always comprises a blood test followed by a doctor consultation and never just a sequence of blood tests or even a solitary diet service, and a specialist nurse service always comprises a specialist nurse consultation and never just a telephone call (these structures were refuted while the *Can_include* relation was still based around the idea of successive 'visits'). If an activity does not include activities of the types Dr Cons and Blood Test, it cannot really be an Initial Visit. The way this essential structure was recorded was through the relation *Comprises*: if $t1$ *Comprises* $t2$, then an activity of type $t1$ cannot be complete until an activity of type $t2$ has been embedded within it and completed. The first invariant confirms the obvious: that all pairs of types in *Comprises* are also in *Can_include*. *Requires* is a function from *Types* to a graphs over *Types*. Thus *Requires*($t1$) would return a particular set of pairs of members of *Types*. The purpose of this structure is to store required orders of types of activities: not only is it necessary that any activity of type Initial Visit *Includes* both a Blood Test and a Dr Cons before it can be completed, but the Blood Test must be *Before* the Dr Cons. We could represent this last observation by including the triple (Initial Visit, Blood Test, Dr Cons) when we specialise the *Requires* function. The last invariant above insists that any pair of types required to be ordered in a particular way with respect to a 'parent' type (from the function's domain) must both be in the relation *Comprises* with that parent type.

The intended use of these two structures is explained further through the formal definition of their interactions with the operational state components of the theory. The class that incorporated *Comprises* and *Requires* into the theory used the following invariants:

((*im ActType*) *Complete*) $\lhd$ *Comprises* $\subseteq$ *ActType* $\circ$ (*Complete* $\lhd$ *Includes*) $\circ$ *ActType*$^{-1}$

$\forall a$: *Complete* • *Requires*(*ActType*(a)) $\subseteq$ *ActType*$^{\circ}$(((*im Includes*) {a}) $\lhd$ *Before*)$^{\circ}$*ActType*$^{-1}$

which define in formal terms the properties described in the previous paragraph.

Comprises and *Requires* were used when an activity was started. If the type of the activity was in the relation *Comprises* with a number of other types, then activities of those other types were created and embedded (in the appropriate order as specified by *Requires*) in newly commenced activity. This reflected the 'automatic' nature of the creation of these activities - no one requests a blood test at the initial visit: it 'automatically' happens as it is assumed by the clinic nurse that it will be required. The way that the theory ensured that the operation had the desired effect was through the use of a very complex precondition[xv].

Although it was considered at the time that some form of structure such as that described was desirable, the particular structure described was refuted quite readily. It might be considered very important, or even declared as necessary, for a blood test to have been carried out before the doctor sees the patient, but if the blood analysis machine in the day centre has broken down or the lab technician has not shown up the doctor consultation will still take place (though admittedly without the recommended level of information) as both the doctor's and the patient's time is too valuable to have them wait until the blood

[xv] one component of which was:
[[t: (*im Comprises*) *ActType*(a)]] •
(*A*1 = {b: (*im During*) {a} | *ActType*(b) ∈ *Dom* (*Requires*(*ActType*(a))}) ∧
(*A*2 = {b: (*im During*) {a} | *ActType*(b) ∈ *Cod* (*Requires*(*ActType*(a))}) ⇒ *CatAct.Embed*(t,*A*1,*A*2,a)!

test has been performed. This does not represent a refutation of the structure *Requires* as it appears in the theory, but intended specialisations (to represent the case above for example) have been refuted. It seemed that there were very few elements that could be put in a specialisation of the *Requires* function such that that specialisation was not to be refuted.

This case with the blood test also refutes desired specialisations of the *Comprises* graph. Another desired specialisation is for Doctor Care to comprise Initial Visit. Here too we can find refutative examples. Sometimes a patient will see the specialist nurse or dietitian before the doctor - especially when there is a long waiting list for doctor appointments: if the patient dies or moves away then we have a completed Diabetes care activity that has never had an initial visit. The same is true if the patient is admitted into hospital suffering from a complication of hitherto undiagnosed diabetes - they will be seen by the specialist nurse, and maybe a doctor so that they can be 'stabilised': if the patient has been taken ill while visiting the area and does not live near the hospital they might register with a diabetes clinic where they live, and never be seen by staff from the Endocrine directorate again.

Only two members of the structure were discovered that were relevant to the DEDC. Firstly a blood test consists of taking blood followed by its analysis - both are necessary and the taking of the blood must precede its analysis. Secondly an Initial Visit must include a doctor consultation - otherwise it is not an initial visit. As we disposed of the visit concept (as discussed above), this second member was excluded. A replacement member which would say that Doctor Care *Comprises* Init Dr Cons might be acceptable, but in the Diabetes Clinic at the Medway hospital, even this assertion is often refuted as that centre uses 'Shared Care' with the General Practitioner - the doctor sometimes never seeing the patient. As the DEDC has started to investigate and promote 'Shared Care' it is considered that cases such as those seen at the Medway hospital will be observed at St Thomas'. In short, there is very little to be gained from incorporating the structures *Comprises* and *Requires* into the theory, and much to be lost in terms of 'semantic overhead': too much complexity in the theory will make it less comprehensible and less useful.

In this case some properties of a proposed theory have been refuted, or at least so many useful specialisations of the theory have been refuted as to render the properties useless. The 'solution' to this problem was to abandon the graph *Comprises* and the structure *Requires* and not replace them with anything else. That there are properties of the domain at least informally or intuitively similar to those described seems likely. After all, most Diabetes Care episodes include an initial doctor consultation, and it is true to say that doctor consultations are almost always preceded by a blood test which are rarely explicitly requested. It seems that medicine is so complex that most general rules of this nature that can be found will either fail, or be of no use in our understanding. We should continue looking for such properties in the domain however, as until we find them some fundamental aspects of medicine have escaped us.

The proposed rules associated with *Comprises* and *Requires* would be more appropriate to a stochastic as opposed to a logical model. Structures similar to those described might be useful in the definition of Care Profiles and Care Protocols, but in this they are not, to use Jackson's terminology [Jackson93] 'indicative' properties of the current domain, rather 'optative' properties of some future imagined one, and so have no place in this particular theory see Sections 13.3 & 14.7 and Appendix 6 for a discussion and example of such hypothetical domain descriptions).

9.5.5 *TypeGuide*: A Replacement for *Can_include*

The representation of *Can_include* as a graph was sufficient to constrain possible values of *Includes* in models of the theory when *During* was a tree: when any activity had at most one parent. As we saw in

Section 9.3.4, it is unrealistic to suppose that *During* is a tree, and we changed its representation to a graph. If we kept *Can_include* in the theory as a graph we would not be able to distinguish between: the case where an activity of a certain type might possibly be *During* any of a number of activities of certain types but not more than one at a time, and the case where an activity of a certain type might be *During* a number of other activities of specific types. A Blood Test might be *During* an activity of type Dr Care, or DSN Care but never both at the same time. An activity of type Diabetic Obstetrics Care would be *During* two activities of type Diabetes Care and Obstetrics Care at the same time, but never *During* only one. To distinguish between these cases we need a more sophisticated structure. The state component used for this purpose in the final theory is called *TypeGuide* and has the following type declaration:

T 11: $TypeGuide: TGrouper \rightarrow (Types \leftrightarrow Types)$

TGrouper is just a collection of markers that enable us to aggregate pairs of types in different ways. For each member of *TGrouper*, *tg*, *TypeGuide*(*tg*) returns a possible structure of types. If we take an activity a of type t and call its parents A_p, there must be a member of *TGrouper*, *tg*, such that the set of types of the parent activities (call this T_p) will be equal to (*im TypeGuide*(*tg*)) {*t*}. To distinguish between the cases above we would have the following as members of the specialisation of *TypeGuide*:

(*tg*1, *Blood Test*, *Dr Care*)
(*tg*1, *Dr Cons*, *Dr Care*)
(*tg*1,...)
(*tg*2, *Blood Test*, *DSN Care*)
(*tg*2,...)
(*tg*3, *Diabetic Obstetrics Care*, *Diabetes Care*)
(*tg*3, *Diabetic Obstetrics Care*, *Obstetrics Care*)

To show how this works, we can consider possible parents of a blood test and of an instance of Diabetic Obstetrics Care. Suppose an activity, *a*1, of type Blood Test only has one parent, *a*2 of type Dr Care. This is allowed as there is a *tg* such that (*im TypeGuide*(*tg*)) {Blood Test} = {Dr Care} - that *tg* is *tg*1. Similarly a Blood Test can be part of an instance of DSN Care, *a*3. Imagine now that *a*1 (a Blood Test) has two parents: *a*2 of type Dr Care and *a*3 of type DSN Care. There is no member of *TGrouper* such that (*im TypeGuide*(*tg*)) {Blood Test} = {Dr Care, DSN Care} in the specialisation of *TypeGuide* we are using, so such a model would be impossible. Now consider the case of *a*4, an instance of Diabetic Obstetrics Care. If we examine the values of the specialisation of *TypeGuide*, we see that there is only one possible *tg* such that (*im TypeGuide*(*tg*)) {Diabetic Obstetrics Care} is not the empty set - this is *tg*3. Now (*im TypeGuide*(*tg*3)) {Diabetic Obstetrics Care} = {Diabetes Care, Obstetrics Care}, so only parents with types Diabetes Care and Obstetrics Care respectively are permissible.

It is interesting to note how a comparatively simple change in the early structure of the theory can result in such a large and complex change in a subsequent invariant: this is inevitably the case as a change in the most basic and primitive classes will carry implications for all the subsequent classes that are either refinements or compositions of them. This is something that was borne in mind when constructing the theory and care was taken to ensure that the most basic classes were those considered to be the most stable. In this way unnecessary work was avoided, and the number of classes that needed regular reworking was minimised.

9.5.6 Other Subsets of *Types*

There are a number of subsets of types that have not been discussed so far. In addition to *HomeTypes*, the sets *Access*, *Unplanned*, *Bookable*, and *PatReq*.

Access is a set of types, activities of which can be created from outside the home organisation and embedded in no other activity. Examples of these are Diabetes Care which can be requested by a GP, another hospital or the Accident and Emergency department, or DECS care which can be requested by a GP irrespective of whether the patient is registered with the day centre. The operation ***.Create*** creates 'orphan' activities, and as such can only be invoked to produce an activity of a type represented in *Access*.

An activity of a type that is a member of the set *Unplanned* can be created as a member of *Proceed*, skipping out the *Request* stage. Examples of such types are Dr Telephone (which happens spontaneously: the person instigating the phone call does not put in a request to do so before making the call), and DSN Cons which might take place without being planned if the patient 'drops in' to see the specialist nurse without having made a booking. The operation ***.SuddenStart*** is the one that creates proceeding activities in this way, and as such can only be invoked to produce an activity of a type represented in *Unplanned*. Note that although an unplanned activity cannot be created as a request, it will become one if it is suspended: for this reason there is no invariant forbidding an unplanned activity from being a member of *Request*.

An activity of a type that is a member of Bookable can be booked: that is it is created as a request and scheduled to take place at a future specified time. Examples of such types are Dr Cons and DECS Cons, both of which can be, and generally are, booked in advance. This subset is considered in more depth later on in section 10.5.2.

An activity of a type that is a member of *PatReq* needs a patient to be present before it can start. This set is discussed in more detail in the next section.

9.5.7 Conclusion

The introduction of *Can_include* means that we can start to construct specialisations of the theory that apply only to one medical domain. An early specialisation that applied to the Diabetes and Endocrine Day Centre was refuted as it could not express the relative autonomy of the specialist nurses and paramedics. The new specialisation of the theory deals with this by having the types DSN Care and Diet Care as siblings rather than children (using the family tree metaphor over the graph *Can_include*) of Dr Care.

Two further specialisation state components, *Comprises* and *Requires*, were considered. *Comprises* specifies what an activity has to include (in terms of types of component activities) if it is ever to be completed. *Requires* specified any ordering (via *Before* or *After*) that was required of the comprised activities. It transpired that there were so few useful specialisations that were not refuted that these quantities were almost useless, and to avoid cluttering the theory were abandoned.

When *During* was changed from a tree to a graph, *Can_include* became insufficient to be able to constrain activity structures of models of the theory in the desired manner. For this reason it was replaced by a more complex structure called *TypeGuide*.

9.6: Conclusion

The classes we have investigated so far represent the foundations of the general theory of the medical process that has been developed over the course of the project. Although we have not really described anything particularly medical such as patients, doctors, medical records, referrals, and the like, we will see that with a satisfactory bedrock to the theory these concepts can easily be 'added on afterwards'. We have reason to believe that the theory defined so far is a foundational one in that what it says is very

abstract and very general (and in fact, prior to specialisation, could apply to many domain areas that are nothing to do with medicine - see Section 14.7 for a discussion of the possible use of the theory to represent other service processes). We have no justification in believing that the theory so far is satisfactory or correct *per se*, only that it is more correct after each refutation / reconstruction cycle than before. Thus it is more accurate to represent *During* as a graph than as a tree, but we should not imagine that because we have improved our representation of the domain we have somehow arrived at the perfect representation (in fact it is extremely doubtful that there is any such thing). The use of the scientific method means that we have some form of yardstick of relative goodness of description, and we should strive to find (among other things) better and bolder theories as measured by that yardstick. The use of the formal notation helps us in this quest by being 'easier to measure'.

The following table lists the original properties and components of the theory, what refuted them, and what replaced them. In this way we can get an overview of the improvements made to the theory as explained so far.

Original Property	**Refutation / Reason for Abandonment**	**New Property**	**Discussed in Section**
Rigid activity life cycle: *Request* → *Proceed* → *Complete*.	Discovery of emergency or unplanned activities, and of importance of temporary interruption of activities.	More flexible activity life cycle through introduction of ***.SuddenStart*** and ***.Suspend***.	9.2.7 & 9.2.8
Before interpreted as ordering required by usage of 'non-shareable resources'.	Blood Test is ordered, but shares no non-shareable resources.	*Before* interpreted as medically meaningful ordering, pertinent to only one patient at a time.	9.3.3
During represented as a tree.	Diabetic Pregnancy Care and Diabetic Shared Care activities have multiple 'parents'.	Re-definition of *During* as a graph.	9.3.4
DSN Care and Diet Care (and other paramedic care) types subsidiary to 'Visit' type in specialisation of theory to DEDC.	Paramedics enjoy a great deal of professional autonomy, and often work in 'parallel' with doctor visits.	New specialisation of theory where *Can_include* has DSN Care, Diet Care and Dr Care as siblings, and disposes of visit entirely.	9.5.3
Comprises and *Requires* define what are necessary components of activities.	Almost all specialisations were refuted - medicine is too varied and unpredictable to be restricted like this.	*Comprises* and *Requires* abandoned and not replaced.	9.5.4
Can_include defined as a graph.	Insufficient to constrain *During* when redefined as a graph.	*Can_include* replaced by *TypeGuide*, a more sophisticated structure.	9.5.5

Table 3-1 Summary of properties that were refuted, the refutations, and the subsequent improvements to the theory

We have discussed seven classes of the theory thus far. The three classes ActClass1, ActClass2, and ActClass3 are successive refinements introducing more state components and operations over those state components. TypeClass1 introduces *Types* which is composed with the activity classes to give ATClass1 which describes interactions between *Activities* and *Types*. TypeClass2 is a refinement of TypeClass1, and is composed with ATClass1 to give ATClass2 which describes the interaction between the graph over *Activities* called *Includes* and the structure over *Types* called *TypeGuide*. The formal theory as presented in Appendix 2 introduces *Types* in a class after the one that introduces Patients. The informal presentation contained in the body of the thesis introduces *Types* before Patients as *Activities*, *Types*, and their interactions are more fundamental (as far as the theory is concerned) than Patients. The following figure illustrates the refinement & composition hierarchy that has been introduced so far

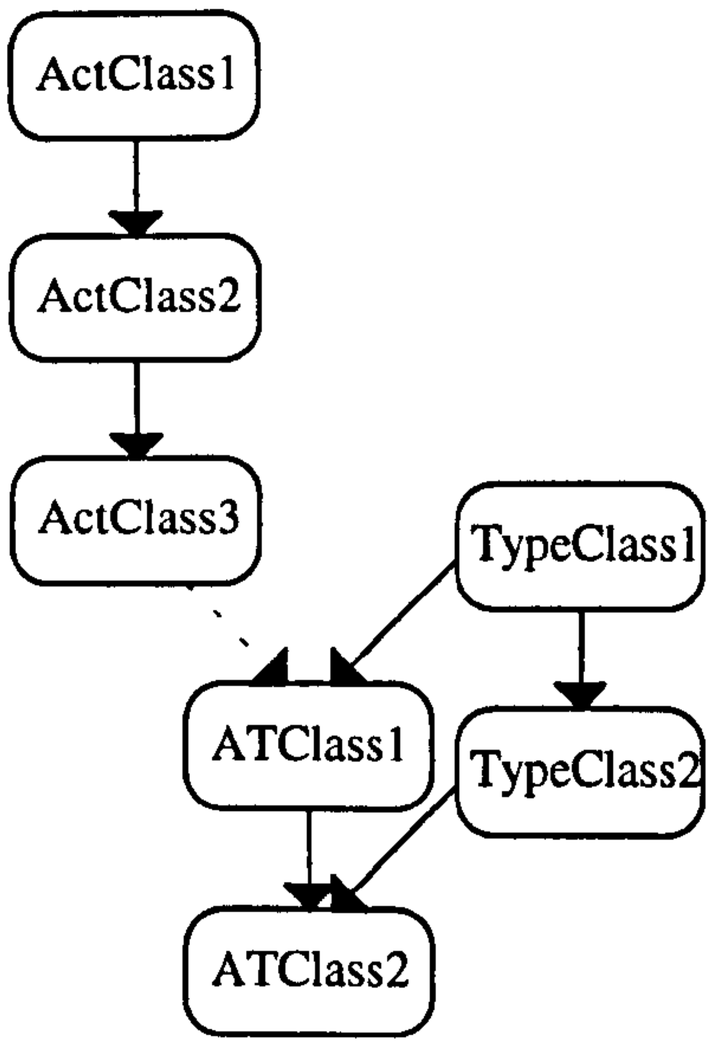


Figure 3-12: Refinement and composition diagram for the seven classes described so far.

The dotted line between ActClass3 and ATClass1 indicates the presence in the formal theory of intermediate classes that have not been explained in the formal text. These are PatientClass and APClass which introduce Patients and are described in the next chapter.

Chapter 10: The Domain Theory II

10.1 Introduction

Now we have explored the underlying structure of the medical process as described by the domain theory, we can embellish it to make models of the theory appear more 'realistic'. For example, we have not yet spoken about patients, clinicians, appointments or medical records: omissions that must all be addressed. In this chapter the way in which these different components interact with the structure created so far is presented and discussed.

The nature of the presentation is similar to that of the last chapter: an initial property of the theory is presented along with its formal definition, invariants and / or operations, any refuting examples explained, and the new (unrefuted) property given.

One of the most important changes recorded in this chapter is that relating to the creating agent for a child activity. The change at this stage was not a simple case of refutation of an initial property and a straightforward reconstruction of the theory - a major conceptual change was required before the theory could be designed that was not unfeasibly complex and was not refuted. This major conceptual change has been called a 'paradigm change' after the philosopher Kuhn [Kuhn70]. This is presented in Section 10.3: Cause and Effect.

10.2 Patients

10.2.1 The Introduction of Patients

So far we have not said anything about patients - a gross omission in any healthcare system - the patient is, after all what the process is all for. We should not be totally shocked to find that the patient is introduced halfway through the presentation of the theory because of the particular domain we are looking at. We are concentrating on the operational essence of the medical process - this is the activity, or interaction between patient and health care professional. A patient can be though of as a way of 'aggregating' these activities so that they have some relevance to the purpose of the enterprise. A patient experiences sequence of activities as they pertain to him or her: this sequence (and what the constituent activities are themselves comprised of) is the concrete manifestation of medical care for a patient. A health care professional experiences a sequence of activities that he or she conducts or at least takes part in: this sequence is the concrete manifestation of the care that is delivered by that clinician. The clinic is an interaction between patient and health care professional which takes place in individual activities. Thus, if we want to understand the essence of the clinic, we do not want to concentrate on either the patient, or the clinician, but on the interaction between the two - the activity. This is what was done in the development of the theory - patients and clinicians being almost 'properties' of the more abstract activity, and certainly not more fundamental concepts.

In spite of this contention that *Activities* is a concept closer to the essence of the medical process than the patient or the clinician (though these too are, of course, absolute requirements), we do not need to refer to the basic set when we are defining the class that will deal with patients - we use the technique of the separation of concerns again. Thus we say that

T4: *Patients*: *Set*[P].

We also represent as a state component those patients that are currently present in the organisation. This set, a subset of patients, is called *PatPres*, where

T6: $PatPres$: Set[P]

also, and

I13: $PatPres \subseteq Patients$.

The theory then defines operations that create (and remove) patients, and represent their arrival at and departure from the organisation. The operation that creates a patient does not represent that patient's birth, but rather their introduction to the home organisation: the act of turning a person into a patient. There are many operations and state components such as these associated with Patients that do not really represent anything in the 'physical' world: there is no physical change to the person, or any aspect of the clinic, when that person has bestowed upon them. The state of some abstract perception of the world shared by the medical personnel in the organisation does change however, and this is what we are effectively theorising about, having no desire or ability to gain access to any 'real' world that might exist independently of these perceptions. The question of what precisely we are constructing theories and models of is examined more closely in the section 13.3.

10.2.2 *ActSubject* - a Definitional State Component

As was said in the previous section, a patient is regarded almost as a property of an activity: a patient is the 'target' or 'subject' of the activity. An activity is only ever targeted at one patient however, and always has such a target - there is no such thing as a 'medically meaningful encounter between a patient and some representative of medical care', even when that encounter is only hypothetical and planned (as in the case of a member of *Request*), without a patient being specified. This assertion is described in the theory with the type declaration

T5: $ActSubject$: $Activities \rightarrow Patients$.

That is, all activities have a subject. This type declaration is partly an assertion and partly a definition of what an activity is. For example, preparing an operating theatre at the beginning of the day, taking dinner to a ward, or fixing the hospital lifts are all excluded from consideration as medical activities because those procedures not being targeted at one identifiable patient: this observation helps us to understand the intended interpretation of the set *Activities*. It becomes an assertion when we find examples of activities that we would want to consider as such, but seem to have more than one patient as the subject.

A case in point is the education sessions held by specialist nurses. The majority of the day centre's patients are NIDDMs: these do not need to be shown how to inject themselves with insulin or given (much) detailed counselling as to the nature of their condition, but rather given an introduction to the issues that they ought to take more care over, and provided with a general education as to the nature of diabetes. This education is delivered in the form of two two-hour long 'patient education sessions'. These sessions are provided by a team of paramedics to a number (generally six) of newly diagnosed NIDDM patients. At first glance, activities of the type 'patient education session' would appear to refute the functional nature of *ActSubject*. The way this was dealt with was to keep to the original definition and say that observable phenomena such as the Patient Education Session were in fact an aggregation of separate medical activities that took place at the same time in the same place, and run by the same clinicians. This change in the way some apparent activities were perceived meant that a later invariant, that a clinician could only be tending to one patient at a time was thus refuted - see Section 10.3.6 for a presentation of this.

10.2.3 The Creation of Patient Specific Activity Structures

Not only do we insist that each activity applies to only one patient, but that any other activity that is in the same 'family' must be targeted at the same patient also. This seems to make sense - Diabetes Care for Mr Smith would not consist of, among other things, a consultation between a doctor and Mrs Jones. We have already discussed this property of the relations *Includes* and *After* in Section 9.3.3 where we saw the invariant

114: $ActSubject \circ (Includes \cup After) \circ ActSubject^{-1} \subseteq id[Patients]$.

Again, this is partly definitional, and partly an assertion. The definitional part we have already seen (in the section mentioned above), but if we look around, we can find activity structures that seem to refute the invariant so in that sense the invariant might be an assertion. One refutative example found recently was activities associated with pregnancy. It is reasonable to think of pregnancy care as consisting of care for the mother before the delivery (*pre-partum*), and of the mother and baby immediately after the delivery (*post-partum*). This is clearly a special case, but has not been addressed by the theory to date, whether by re-casting the invariant or representing the birth in an artificial way so as to comply with the existing theory (as we did with the Patient Education Session).

10.2.4 Patient Presence: its Necessity and Cardinality

It might be thought that activities that cannot include any others by virtue of the relation *Can_include* (or its derivative *TypeGuide*) always need the patient to be present if they are to start. This is not so. A refuting type of activity is Blood Analysis which does not need patient presence, or at least does not need the whole patient to be present - it is sufficient for the patient to be represented as a vial of blood. There are also types of activity that may or may not have a patient present at them (the Blood Analysis activity never has a patient present) - an example of this is was discovered at the beginning of the project in the form of the types of activity observed in the general haematology clinic at St Thomas'. Many people who suffer from coronary problems need Warfarin to thin their blood: they will always need it, and they will almost certainly need the same amount. In this case it is not necessary for the doctor to always see the patient: sometimes a 'postal encounter' will take place where the doctor writes out a prescription for warfarin and sends it to the patient. Occasionally the patient does attend the clinic if there is something bothering them about their treatment and they want to see the doctor.

In spite of all these cases, there are many types of activity where the activity cannot start unless the patient is present. These types are recorded in the subset of *Types* called *PatReq*. With activities of these types, we say in the theory that the operation *.Create* cannot start unless the patient has not only arrived at the clinic, but is present at the activity about to start - of course, the patient that is present at the activity must be the same as the patient who is its subject.

A more general rule that we might think about is that restricting patient presence to one activity. An early version of the theory did just this, using a state component called *PatAct* which was defined to be

$PatAct: Patients \nrightarrow Activities$.

whereupon we could say that

$ActSubject \circ PatAct \subseteq Id[Patients]$.

PatAct thus records which activity a given patient is currently present at: the invariant states that the patient present at the activity is also its subject. This property of the theory was refuted by the example of the type of activity that is called (in the specialisation of the theory to the DEDC, and for want of a more formal name) the 'Dr Pop-in'. The chiropodist, as a paramedic, is not formally qualified to dispense drugs. Occasionally, however, she will feel that the most appropriate treatment for a patient's foot related problems is for that patient to be prescribed a particular curse of drugs. When this happens the chiropodist finds the patient's usual doctor and asks them to see the patient, agree the therapy, and sign the prescription form for the drug. This activity does not take long, as the assumption is that the chiropodist will act appropriately with the doctor only ratifying her decision. The important thing to note here is that the patient must be present for the chiropodist encounter to start, and for the Dr Pop-in to start. We can see here then that if the patient is present at the Chiropodist Cons, then they will be present at the Dr Pop-in, thus refuting the earlier insistence that a patient could only be present at one activity at a time. Other examples of patients being present at multiple activities simultaneously occur in a surgical operation (that requires the presence of a patient to start) which might consist of sub-components (that also require the presence of patients to start).

The final theory deals with the problem by allowing patients to be present at a number of activities at a time. The set of activities that have a patient attending them is *ActAtt* (Activities that are Attended). where

T6: $ActAtt: Set[Activities \backslash Complete]$.

All attended activities are so attended by the patient that is their subject - something that is ensured by the operation that assigns patients to activities. A relation that related activities to patients present at them would thus be given by:

$ActAtt \lhd ActSubject$

Although a patient can now be present at a number of activities, those activities must be in the inclusion relation with each other - for example in the example given, instances of Dr Pop-in are always components of instances of Chiropodist Cons. The way we say this is to use the following predicate as an invariant:

I15: $\forall p: Patients \bullet ((im\ (ActAtt \lhd ActSubject)^{-1})\ \{p\})^2 \subseteq During^* \cup Includes^*$

This says that any pair of activities that are attended by a patient p, must either be in an ancestral relation, or a descendant relation (or be the same activity). This invariant is satisfied through the preconditions of the operation that assigns patients to activities: ***PatJoin***(*a*,*p*).

The recast theory is not quite as strict as the original, but only the minimum amount of flexibility has been introduced to prevent it from being refuted by the counter examples described above. We could have said that a patient could be at a number of activities and left it at that - this would be difficult to refute, but precisely because it is difficult to refute, it is not very informative. Popper's injunction that we should strive for theories that are increasingly more falsifiable has been used to guide decisions taken with regard to patient presence, just as it has guided the entire analysis.

10.2.5 Conclusion

Patients, although essentially the purpose of all health care, are not the most basic concept at an operational level. This status should rather be accorded to the interaction between patients and clinicians, in other words the patient activity. Patients do need to be introduced, and this was done in this section. *ActSubject*, a total function from *Activities* to *Patients* is discussed, and explained is part of the definition of an activity, and partly an assertion about an activity. The fact that the assertion might be refuted leads us to change slightly our interpretation of what an activity is, so as to observe the definition. In this case, the Patient Education Session must be interpreted as a number of parallel activities that start and finish at the same time, run by the same clinician: this interpretation is not the most intuitive, but it observes the type declaration of *ActSubject*.

The assertion that any 'activity structure' is associated with exactly one patient is similarly partly definitional of *Includes* and *Before*, and partly assertive. Here, the assertion is challenged by activities associated with child-birth, a problem that is not yet resolved.

Originally, a patient could only be physically present at one activity at a time. This theorem was refuted by the example of the Dr Pop-in activity leading to the property being slightly relaxed.

10.3 Cause and Effect

10.3.1 Introduction

We have so far looked at a number of constraints governing what the possible activity structures of models of the theory will look like. We have not investigated when new activities are created. As far as the theory is concerned so far new activities might spontaneously come into existence, so long as they are of a type that allows them to be children of there parent activities. One of the early contentions held by the author was that any activity must be created as a result of another. Thus, as a result of a consultation with the doctor, a specialist nurse activity might be created (if the doctor decided that the patient needed to see the nurse), booked, and run at the appropriate time. At the same consultation the doctor might decide that the patient needed to come back to see him or her again in six months time for a followup consultation. In a sense, we can say that both the specialist nurse consultation and the followup doctor consultation were created as a result of, or even by, the initial doctor consultation. There are constraints that seem to be demonstrated in this area in the domain. For example, although a blood test might be created as a result of a doctor consultation, the reverse is not observed - a blood test would never create a doctor consultation (or so it was believed at the time). Rather the results of the blood test would be reviewed by a doctor at a future consultation, and it would thus be at that doctor consultation that the required future activities were decided on. Similarly, as a result of a specialist nurse consultation, a followup specialist nurse consultation might be created. It is never observed in the day centre that a specialist nurse consultation is created by a chiropodist consultation.

There are clearly some rules governing when an activity can be created, who can create it, and as a result of which activity: what these were was not clear at the beginning of the project. It was decided then that this area of the domain, the issue of cause and effect, was worthy of investigation as it had the potential to add a great deal of rich and relevant structure to the theory.

10.3.2 *InLoco* as a Two-Place Relation

An early class tried to address the problem through the introduction of a number of new graphs and other structures over existing state components. One of these was the graph *CreatedBy*, defined in the type declaration:

CreatedBy: *Activities* $\twoheadrightarrow$ *Activities*.

The intended interpretation of this function is as a record of which activities gave rise to which others. Thus if the pair ($a2$,$a1$) is a member of *CreatedBy*, the activity $a1$ gave rise to the activity $a2$. As in most of our other graphs, this structure is directed and acyclic:

$CreatedBy^{+} \cap id[Activities] = \varnothing$.

The function *CreatedBy* was updated every time a new activity was created and embedded in another: when operations of this sort were invoked, the creating activity had to be supplied as one of the required arguments.

The behaviour that this early version of the theory attempted to represent was as follows. An activity can always create any of its allowable 'children': an activity of type Diabetes Care can give rise to instances of Dr Care, DSN Care or Diet Care among others; an activity of type DSN Care can give rise to activities of types DSN Cons or DSN Telephone. An activity can also, under certain circumstances, give rise to a child of one of its ancestor activities. For example, as a result of a consultation, a doctor might want to create a DSN Care activity whereupon we would have an activity of type DSN Care *CreatedBy* an activity of type Dr Cons. Although it is not true that Dr Cons *Can_include* DSN Care (at least not in the specialisation we are using for the DEDC), it might be true that the Dr Cons activity is a child of a Dr Care activity, and the Dr Care activity a child of a Diabetes Care activity which allows as possible component activities instances of (among others) DSN Care. In this case then an activity has not given rise to a child activity, but rather a child of an ancestor (in this case the appropriate activity of type Diabetes Care).

We do not want any activity to be able to give rise to any possible children of all ancestors - a Blood Test activity would not give rise to a Diet Care activity for example. We constrain possible events of this type using a graph over types called *InLoco*. This is defined in the theory in the following way:

InLoco: *Types* $\leftrightarrow$ *Types*

$InLoco \subseteq (Can\_include^{+})^{-1}$

If one type of activity, $t1$, is *InLoco* another, $t2$, then an activity of type $t1$ can act as if it were one of its ancestral activities, of type $t2$, and create child activities of that ancestor (i*n loco* is Latin for 'in place of' and is usually used to describe the relationship between a teacher and a pupil - the teacher acting *in loco parentis*). The invariant says that if type $t1$ is *InLoco* type $t2$ then $t2$ has $t1$ as a child through the relation *Can_include* (ie using the 'family tree' metaphor for *Can_include* as we did for the graph over *Activities*, *Includes*). The use of *InLoco* is defined in the invariant linking it to *CreatedBy*:

$(a,b) \in CreatedBy \Rightarrow (a,b) \in During \vee (\exists\ c: Activities \backslash \{a,b\} \bullet (a,c) \in During \wedge (b,c) \in During^{+} \wedge (ActType(b),ActType(c)) \in InLoco)$.

This invariant is structured in a similar way to the informal argument used to describe allowable *CreatedBy* values presented above. If a is CreatedBy b, then one of two things is true: either a is *During* b,

or a has a parent c which is an ancestor of b, and activities of the same type as b are capable of acting *InLoco* activities of the same type as c (At this stage in the theory's development *During* was still a partial function).

This then was the (somewhat complex) structure that was proposed as a theory of the domain of interest. This property of an early theory was abandoned as it was not sufficiently semantically rich to represent the behaviour that is actually seen in the DEDC. Notably, this collection of theorems cannot reflect the selectivity of the creation of activities by different professionals: an activity might be able to give rise to some components of an ancestral activity, but not any. For example, a Diet Cons activity might be able to give rise to a DSN Care activity, but not a Ophthalmologist Care activity.

10.3.3 *InLoco* as a Three Place Relation

This asymmetry was dealt with through the re-definition of *InLoco* as a three place relation:

$InLoco$: $Types \leftrightarrow (Types \leftrightarrow Types)$

If we have a triple ($t1$,$t2$,$t3$) in *InLoco*, we know that an activity of type $t1$ can embed an activity of type $t3$ in an activity of type $t2$, subject to all the usual constraints. To indicate that an activity of type Diet Cons can give rise to an activity of type DSN Care as a part of Diabetes Care, we would put the triple (Diet Cons, Diabetes Care, DSN Care) in the relation. Clearly in this case the pair (Diabetes Care, DSN Care) must be (and is) a member of *Can_include*. In fact we can specify the invariant:

$Cod(InLoco) \subseteq Can\_include$

which asserts the general case. The activities of the first type in the triple must be possible descendants of activities in the second type (barring other constraining factors) which is asserted by the invariant:

$\forall a: Dom(InLoco) \bullet \forall b: Dom(Im\ InLoco\ \{a\}) \bullet (b,a) \in Can\_include^+$.

The interaction between this specialisation state component and the operational state component *InLoco* is given in the following invariant which is similar in intent to that proposed when *InLoco* was a two place relation:

$(a,b) \in CreatedBy \Rightarrow (a,b) \in During \vee (\exists c: Activities \backslash \{a,b\} \bullet (b,c) \in During^+ \wedge (ActType(b), ActType(c), ActType(a)) \in InLoco$.

This means that if a is *CreatedBy* b then either a must be *During* b, or there is a third activity c that is an ancestor of b and where the types of b, c, and a are in the triple *InLoco*. Because of the invariant over *InLoco*, from this we can deduce that a is a child of c. Using the family analogy still further, we can say that if a is *CreatedBy* b, then a is either a child of b, or a child of an ancestor of b.

The new collection of theorems although working more effectively than the old, was refuted and abandoned. The counter-example in this case was the creation of specialist nurse consultations. The doctor might, during a Dr Cons activity, decide that the patient needs to receive DSN Care which can be accommodated under the existing theory. The doctor might also book the patient in to see the specialist nurse him or herself, or at least the clinic clerk will do on behalf of the doctor. Here a Dr Cons activity has given rise to a DSN Cons request: DSN Cons is not a child of an ancestor of Dr Cons - it is a 'grandchild'. This example also refutes the earlier two-place *InLoco* relation.

10.3.4 A 'Paradigm Shift'

The new structure was already fairly complex (and this was while *During* was still a function): any alteration of the form of the various state components so as to accommodate the observed refutative example would add still more complexity and the resulting invariants and state components would become incomprehensible. This aspect of the theory did not seem to work, and it was difficult to see how it might. The solution came in the form of a 'paradigm shift', or a challenging of (some of) the underlying assumptions on which the theory was based.

One of the earliest assumptions of this part of the theory was that activities were created by other activities. The goal was, for any activity, to be able to derive a chain of its causative activities. That an activity is caused by another seemed like a good idea - the problem was how to frame this causation in technical terms. That an idea is appealing to an analyst does not mean that it is a useful one with which to understand the domain, and in the case of *InLoco*, the Byzantine nature of the various possible (unrefuted) theorems and structures would indicate that that particular idea was neither particularly useful or insightful. Indeed, looking back through the records of the formal interviews the analyst had with a number of clinicians, no mention was made of the referring activity, only of the referring clinician. The fact that this was not picked up on by the analyst is symptomatic of the 'paradigm trap' discussed in Section 13.3. In short, the appropriate course of action at this point in the development of the theory was to abandon the necessity of recording the creating activity altogether, and instead look at who was able to create which activities and when.

Once it was decided to dispense with the *CreatedBy* state component and its associated ideas and assumptions, several things that were puzzling before were understandable, and capable of being represented. We said that a blood test could not give rise to a doctor consultation, and that if it was to be read, there must be activities that have not finished that will be able to respond to the results of the blood test. In the case of the DEDC this will normally mean that the results of a test are reported back to the current doctor consultation, if the blood test is very rapid, or to the next doctor consultation if the test takes a bit longer. For some conditions however, there will be no followup consultation for the test results to return to. Instead, they are sent to whoever requested them at which point a decision is made whether to create a new activity. If we were still labouring under the 'activity begets activity' paradigm, we would have to invent a new activity type that represented the act of perusing the test results, an instance of which was created by the test activity. If we had defined all these and similar interactions as types of activities, the specialisation to any domain of medicine would become unfeasibly huge. Moreover, we would be forcing the idea of clinical activity to unrealistic extremes - maybe we could think of a doctor's perusal of a patient's medical record, or a computer's monitoring of desired patient recall dates as clinical activities - but to do so would not reflect an intuitive understanding of what kind of thing an activity is. By thinking of the behaviour of the domain in terms of health care professionals creating activities rather than activities creating activities, we can cut down on the number of types and represent what is observed in a more 'reasonable' manner.

10.3.5 The New Theory and *EmbedType*

The new theory disposed of the state components *CreatedBy* and *InLoco* and replaced them with structures controlling what sort of person is capable of creating different types of new activities. In order to do this, the theory has to describe those people who are now the creators of activities - the clinicians. We have

T12: *HCP*: *Set*[P]

where *HCP* stands for Health Care Professional. Health care professionals are also people which is why they are taken from the same carrier set as patients: *HCP* could be taken from a completely new carrier set, however, and not affect the behaviour of models of the theory. Values of *HCP* might include Peter, Jake, Gill, Jill and Sara. Each *HCP* has a clinical type which we might call 'Profession'. We assume here that the set of types of clinicians never changes so we can say that this set is an unchangeable carrier set: Pr. Each clinician has one profession which is recorded via the function *ProfType*, defined as

T 15: $ProfType: HCP \rightarrow Pr.$

Though this may change, we do not specify in the theory how: *ProfType* is a specialisation state component. In the specialisation of the theory to the DEDC, *ProfType* might include the pairs:

(*Peter*, *Doctor*)
(*Jake*, *Doctor*)
(*Gill*, *Dietitian*)
(*Jill*, *Chiropodist*)
(*Sara*, *Diabetic Specialist Nurse*)

Each profession is capable of creating a subset of all those activities that can be created. The way we represent this is through the use of a structure called *EmbedType*. *EmbedType* is a function from a profession to a subset of *TypeGuide*. The type of this function is declared as:

T 16: $EmbedType: Pr \twoheadrightarrow (TGrouper \twoheadrightarrow (Types \leftrightarrow Types)$

where

136: $Cod(EmbedType) \subseteq TypeGuide$

Now suppose that a new activity of type *tc* can be embedded in two activities of types t_{p1} and t_{p2} simultaneously, then there will be some member of *TGrouper*, *tg*, such that

$\{(tg, tc, t_{p1}), (tg, t_c, t_{p2})\}$

is a subset of *TypeGuide*. If a clinician of profession *pr* is allowed to embed a new activity of type t_c in an activity of type t_{p1} when that activity is intended to be a part of two activities of types t_{p1} and t_{p2} simultaneously then we know that the tuple

(pr, tg, tc, t_{p1})

must be an element of *EmbedType*. However, if the clinician is to be able to embed the new activity in an activity of type t_{p1} through recourse to this structure, she must also be able to embed it in an activity of type t_{p2} as the activity must be embedded in both simultaneously. This means that the tuple

(pr, tg, tc, t_{p2})

must also be an element of *EmbedType*. We ensure this in the general case with the invariant

$\forall pr: Pr; tg: TGrouper; t: Types \bullet$
$(im\ EmbedType(pr)(tg))\ \{t\} = \varnothing \vee (im\ EmbedType(pr)(tg))\ \{t\} = (im\ TypeGuide(m))\ \{t\}.$

This means that for any profession, value of *TGrouper* and type, either a clinician of that profession is not allowed to embed an activity of that type in any parent activities of the types specified by *tg*, or she will be

able to embed it in all parent activities of the types specified by *tg* (the invariant is presented somewhat differently in the theory as I37).

The operation to embed a new activity in another now requires the name of the clinician who is the creator to be supplied as an argument. The class in which *EmbedType* is defined is called ATClass4 - the operation to embed a new activity is thus ***ATClass4.Embed***$(A_p,A_b,t_c,hcp\rightarrow a_c)$. A precondition ensures that the operation is only invoked when it will result in a model that is permitted under invariant I37. The operation as it appears in the theory is thus:

AT4.Embed$(A_p,A_b,t_c,hcp\rightarrow a_c)$

Pr97: hcp: *HCP*

Pr98: $\exists tg$: *TGrouper* • (*im ActType*) A_p = (*im EmbedType*(*ProfType*(*hcp*))(*tg*)) $\{t_c\}$

Pr99: ***AT3.Embed***$(A_p,A_b,t_c,hcp\rightarrow a_c)$

.

The latest theory is not particularly difficult to grasp conceptually, but the invariants and preconditions are fairly clumsy. This probably indicates that the same (formal) effect could be achieved through using a different set of set-theoretic constructions. Tidying up the representation of the theory in this way is one of the areas for future work described in Section 14.7 of the conclusion.

We can see that this new theory allows (with the appropriate specialisation) the behaviour observed in the domain that refuted the previous version. Consider a specialisation of the theory which included the quadruple (Doctor, *tg*2, DSN Cons, DSN Care) in *EmbedType*. Suppose we have the following (fragment of a) model of the theory:

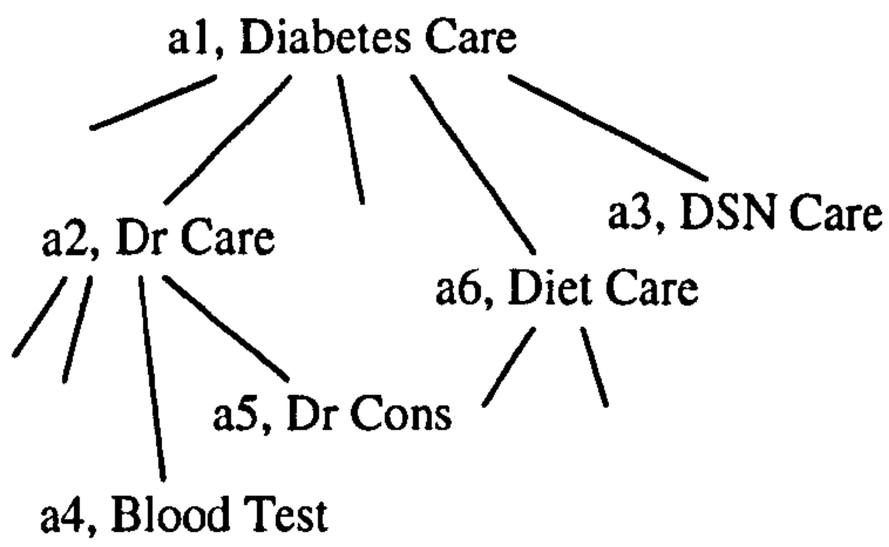


Figure 3-13: Fragment of a model of the theory, specialised to the DEDC.

The links with only one node indicate components that have not been specified. The important relation here is: *a*1 *Includes a*3.

Now, with the model in this state, we can invoke

AT4.Embed$(\{a3\},\varnothing,$ *DSN Cons*, *Peter*$\rightarrow a_c)$,

which has the precondition (on invocation - ie with the variable arguments replaced with values):

$\exists tg$: *TGrouper* • (*im ActType*) $\{a3\}$ = (*im EmbedType*(*ProfType*(*Peter*))(*tg*)) {*DSN Cons*}

The specialisation of the theory to the DEDC contains the following values in the structure *EmbedType*:

...
(*Doctor*, tg_2, *DECS Care*, *Diabetic Care*)
(*Doctor*, tg_2, *MARS Care*, *Diabetic Care*)
(*Doctor*, tg_3, *Followup Dr Cons*, *Dr Care*)
(*Doctor*, tg_4, *DSN Cons*, *DSN Care*)
(*Doctor*, tg_4, *DSN Edcn Session* 1, *DSN Care*)
...

Assuming these (more than) are the only elements of *EmbedType* of concern to us, we can argue as follows.

We know that

(*im ActType*) {*a*3} = {*DSN Care*}

and

ProfType(*Peter*) = *Doctor*.

Thus we are looking for an element from *TGrouper*, *tg*, such that

{*DSN Care*} = (*im EmbedType*(*Doctor*)(*tg*)) {*DSN Cons*}.

If we can find such an element then the precondition is satisfied and the operation is permitted. We can find such an element: it is tg_4. We can see this as follows:

(*im EmbedType*(*Doctor*)(tg_4)) {*DSN Cons*} =
(*im* {(*DSN Cons*, *DSN Care*)} {*DSN Cons*} =
{*DSN Care*}.

Thus the precondition is satisfied and we have the new model:

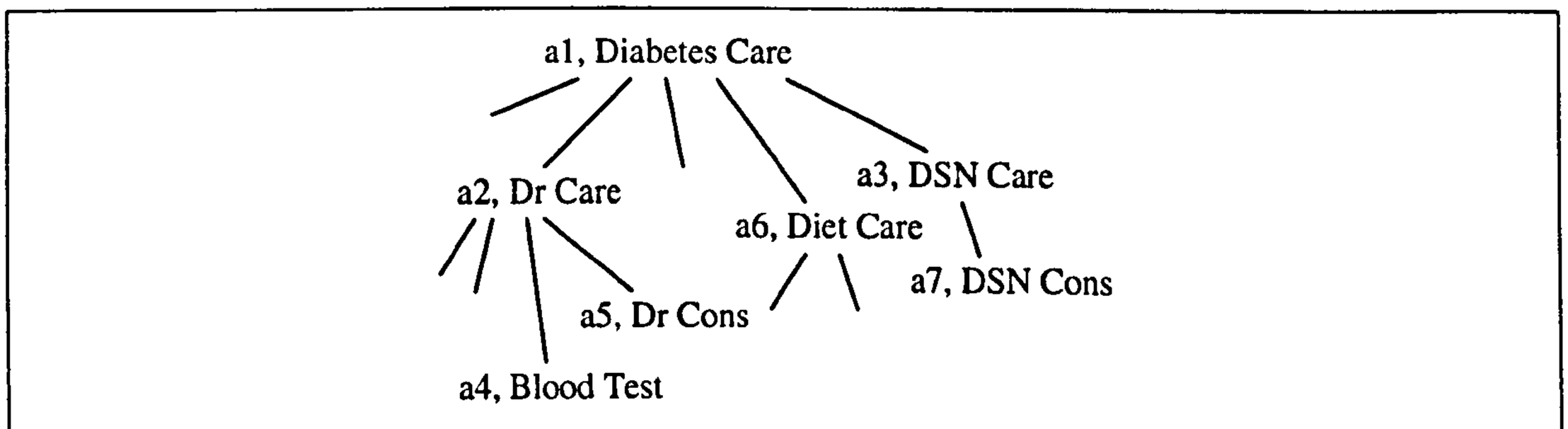


Figure 3-14: Fragment of same model of the theory, after invocation of the operation ***AT4.Embed***({*a*3},∅,DSN Cons,Peter→a_c)

10.3.6 *RunType*

The other specialisation state component that links professions to activity types is *RunType*. This component of the theory records which types of professionals can start which types of activity. For example, a Dr Cons activity can be 'run' - ie started, suspended and completed -only by a Doctor, not by a

Dietitian, Ophthalmologist or any other type of clinician. This is represented in the theory as a total relation between *HomeTypes* and Professions - for every type of activity the home organisation can run there is a professional that belongs to the organisation that can run it. Thus

T14: $RunType: HomeTypes \leftrightarrow Pr.$

The way the theory applies this relation is as follows. Before an internal activity can start (or be suspended or completed), it must be in the domain of the partial function *ActRun*:

T15: $ActRun: In \setminus Complete \nrightarrow HCP.$

This records the health care professional that is currently responsible for running a particular activity. Not any clinician can be associated with any activity through *ActRun* - the invariant

I34: $HCPProf \circ ActRun \circ ActType^{-1} \subseteq RunType$

means that only clinicians who are of a profession that can run the type of activity in question are allowed to be so linked to that activity.

Early on in the project, it was imagined that a clinician could only run one activity at a time. This is untenable however - a clinician might be running one activity and decide that a subsidiary activity was required and thus needed to be run - in this case the clinician would be running two activities. An example of this would be the decision of a doctor, running 'Doctor Care' to create a subsidiary activity 'Doctor Consultation' and run this, concurrently with the first. A new invariant was thus added as follows:

$\forall hcp: Cod(ActRun) \bullet \forall a_1,a_2: (im\ ActRun^{-1})\ \{hcp\} \bullet (a_1,a_2) \in Includes^* \cup During^* \vee a_1 = a_2$

This says that if a clinician is running more than one activity, then any two of those must be in an ancestral relation with each other.

This property was refuted by the same example that caused us to question the nature of the Activity - Patient link: namely the specialist nurses' patient education sessions. We said that an activity is related to one patient only via the function *ActSubject*, and that the Patient Education Session is in fact several activities occurring in parallel. A Specialist Nurse will be present at all of those activities simultaneously, and will in fact be running them all. The previous invariant is falsified by this example, and must be replaced. In the event, it was not clear what an alternative invariant might look like: a decision was taken to say nothing about restrictions on concurrent activities being run by the same clinician. The theory, while not being refuted in this respect, is inevitably much weaker as a consequence of this decision. This is thus considered one of the fruitful areas of further investigation and is presented as such in Section 14.7.

10.3.7 Conclusion

This section has explored the notion that all child activities are created by some sort of 'agent' whose identity and attributes are recorded in models of the theory. Initially it was assumed that one activity was created by another. With this underlying assumption, or paradigm, increasingly complex structures and behaviours were represented in the theory to avoid a succession of refutations. This increasing complexity was not only formal, but conceptual too. No progress was made with the problem until the underlying paradigm was changed, and the creating agent was considered to be a clinician rather that another activity.

Physical presence of clinicians at activities is represented via the function *ActRun*. An activity cannot start until a professional of the correct type is associated with it via the function. Originally a clinician could be associated with only one activity at a time: this theorem was refuted by the example of the patient education session and replaced with a less stringent one.

10.5 Further Enrichments and Enhancements of the Domain Theory

10.5.1 Introduction

It is not the intention of this chapter and the last to explain the whole theory in detail, rather to indicate how it was derived through the theory construction and refutation cycle. However there are some classes of the theory that were arrived at without much reworking. Some of these classes are those that refer to time, and the booking and scheduling of activities. These classes nevertheless represent an important aspect of the theory and are introduced in this section. More detail that is particularly relevant to the clinical domain is the keeping of medical records on patients, and communicating those records to other healthcare professionals.

Further enhancements that might be made to the theory lie in the form of the operations that are invoked to change the state of its models. Little attention has been paid yet to the link between the types of operations that the theory supports and the type of events that are observed in the domain. This is discussed in this section.

Finally a few words must be said about the representation of boundaries of the particular domain of interest. How might we interpret such things, and what is a boundary in 'reality': this is discussed in the final part of this section below.

10.5.2 Time and Booking

Because of the compositional nature of the notation, we can represent time independently of the body of the theory, and then incorporate it when it is needed. This we do through the classes Clock which is refined with Scheduler which in its turn is composed with ATClass4 to produce ATClass5. The details of the class is not important here, being presented with the rest of the theory in the Appendix 2. The reader who is interested in the formal representation of booking systems is referred to the CAVIAR specification which is clearly presented in [Flinn85], and from which some ideas in the theory that are pertinent to booking and scheduling are taken.

The introduction of time into the theory allows us to say when activities are created, when they start and when they stop. This 'time stamping' of activities allows us to 'tighten up' the looseness of the earliest class. We saw in section 9.2.8 that there was nothing to stop an activity from being requested, started, suspended and then cancelled which is a behaviour not observed in the domain. By assigning times to activities, we can distinguish between those members of *Request* that have started and been suspended, and those that have never started. A precondition of the cancel operation is

Pr 128: $(im\ Includes^*)\ \{a\} \cap Dom(ActStart) = \varnothing$

thus preventing the cancellation of any activities that have been suspended, or have descendant activities that have been suspended.

We can also assign times to certain types of activities (specified by the subset of *Types* called *Bookable*) indicating when we would like them to start and stop - this is the way we represent the booking

procedure. The way this works is that an activity that is to be booked is assigned a 'slot' which has a beginning and an end. Each slot belongs to a clinic list, of which there are a fixed number (the names of clinic lists are a specialisation state component). No two slots on the same clinic list can overlap. This is the basis of the booking system in the DEDC, and most other medical domains (in surgery for example, the 'clinic list' would represent an operating theatre).

All activities that are booked and have never started are members of the set *Request*. All members of *Request*, even those that are of types in *Bookable* are not necessarily booked however. This property of the theory allows us to represent waiting lists - a situation where bookable activities are requested, but are not booked until they come to the 'front of the queue'.

Although the development of this class of the theory is not of great interest, it is pertinent to the later integration of the outpatient appointment system with the clinical record system, and into the domain. For details of the behaviour of the domain with regard to bookings, the reader is referred to the formal theory presentation in Appendix 2. Here the behaviour is expressed in terms of such state components as *Slots*, *ActSlots*, *Clist*, and so on.

10.5.3 Information and communication

As was discussed in the last section, before the attempt to give each embedded activity a single creator activity was abandoned, the theory developed undesirable complexity. This was needed to represent the observed behaviours of the domain within the chosen paradigm. Much of this complexity was introduced to deal with the issue of medical communication between professionals. Through discussions with clinicians it was revealed that there were two sorts of 'internal' referral in the DEDC. The first was a referral from the doctor to another paramedic. If the doctor decides (presumably as part of some activity) that the patient needs to see the specialist nurse, an activity of type DSN Care will be created to enable that. The doctor might go so far as to book the patient in for an appointment with the specialist nurse in which case an activity of type DSN Cons would be created as well as one of type DSN Care. If a specialist nurse thinks that a patient ought to see the doctor, or dietitian or chiropodist, she will talk to the professional concerned and it will be as a result of this conversation that a decision is made. If we wanted to represent this second method of internal referral using the assumptions implicit in the 'activity as creator' paradigm, we would have to create a new type of activity to represent this conversation between clinicians. This is what was done in an early version of the theory, there being a special member of *Types* called 'Professional Dialogue' for which there were special rules.

When this way of thinking about the domain was abandoned, the consultation process no longer had to be represented as an activity, but it is nonetheless important and must be incorporated into the theory somehow. Communication needs to be understood if we are to be serious in our efforts to have the theory represent causality - many activities are created as a result of communication between health care professionals inside and outside the clinic.

Although the final theory does not depict 'inter-professional consultations' as explicit state components, it recognises that they nevertheless give rise to activities. Thus if a specialist nurse phones up the chiropodist to discuss the necessity of the patient getting foot care, then although the chiropodist might decide to not see the patient, the act of talking about him or her to the specialist nurse might be considered to represent a form of care: the state of the patient is being considered by a health care professional, and a medical decision is being made on the basis of that consideration. The telephone conversation (or whatever form the communication takes) is thus a sort of referral, and is portrayed as such in the theory.

While the theory does not explicitly represent communications, it does incorporate the content of that communication into its theorems and state components. The content of a patient-related communication is information about that patient, and although communications are not represented as such, information is. We must be careful when talking about information - we do not mean the symbols that are recorded on pieces of paper any more than the set *Patients* meant the written names of the clinic's patients. Information is used in association with the medical record. This record is a representation of the perceived state of the patient's health. It differs from a clinician's hunch or fleeting opinion in that it is recorded somewhere, and is thus part of the organisation's knowledge rather than any particular individual's. As such it is a state component as much as any other. The perception of the state of health of a patient will place constraints (of a possibly intractable nature) on the conduct of the organisation, in just the same way as the registration of a particular patient, the booking of an appointment, or the employment of a clinician. These medical records, being perceived states of health, are depicted in a very abstract manner: information is considered to be a totally unstructured (carrier) set I. In order to link this information with patients and activities we use a set of records -

T30: $Records: Set[\mathsf{R}]$

where each record has an informational content given by *RecCont* (Record Content):

T31: $RecCont: Records \rightarrow \mathsf{I}$.

Whenever something is recorded about a patient, a new member of *Records* is created and associated with a member of I. This is about as crude a representation of patient data as can be devised. This reflects the decision taken at the start of the analysis not to get bogged down in the 'medical knowledge' side of the clinical process (clinical records systems are an interesting area, but are proving to be a barely tractable problem). This issue was discussed earlier in Section 7.3.

Each clinical record has one activity associated with it as defined by the type declaration of RecSource -

T33: $RecSource: Records \rightarrow Activities$.

Note that an activity might have many records associated with it. The quantity

$(im\ RecSource \circ RecCont^{-1})\ \{a\}$

thus gives all information associated with activity a. This is not just the information recorded in the official 'patient notes', but all forms of information that might be associated with the activity, be it in a computerised, typed, written or scribbled form. This reflects the interpretation of the medical record as the organisation's collective perception of the state of health of a particular patient. The reason why we want to allow such flexibility is in recognition of the fact that the introduction of operational computer systems into an organisation will inevitably change the type and manner of the storage of information: we do not want to be in a position where we cannot consider one form of activity information that turns out to be much more important than was imagined. Thus, a telephone conversation is represented by the operation

$ATClass7.ReferInt(a_s,A_p,t_c,hcp,i \rightarrow a_c,r)$

which represents a telephone call which is also a referral (referrals will be discussed in the next sub-section).

Communication that cannot be a referral (for example by a clinician to a type of activity that she cannot ***.Embed***) is not represented explicitly at all. This is because there would be no change in the state of a model of the theory were such an operation to be invoked that distinguishes it from the more general operation ***.NoteTake***. In other words, such an operation would be semantically identical to that latter one. The medical record represents the organisation's perception of the patient's state of health - we are not concerned here how that perception is distributed within the organisation. Thus in sending a message to another health care professional, the clinician augments the record, the source of that augmentation being the activity that was being conducted when the message was written. Once written, the theory says that there is open access to its contents. While this might seem unrealistic, it is not as it stands inaccurate - merely unrefined. The mechanisms for medical data access are liable to be as complex as those for its storage: we want to avoid getting dragged into contemplating the former for the same reason as we want to avoid the latter.

Although this is the limit of consideration of medical records in this statement of the theory, a foundation has been laid that enables future restrictions and enrichments (that will be less generic) to describe data storage, data access, referral and communication in a way which is coherent and consistent with the behaviour of the organisation as described in the body of the theory.

10.5.4 Final Operation Refinements and Followups

The amalgam of activities, patients, clinicians, types, professions, time and 'information' described so far enables models to be constructed that can represent the directorate with a fair degree of 'accuracy'[xvi]. The operations are still too general to enable us to capture important aspects of the behaviour of the medical domain. The most significant operation that creates new activities is ***.Embed***. Although this is a reasonable operation in that with it we can represent the majority of medical event inceptions, it is not one that accurately reflects what is perceived to happen in medicine: doctors do not 'embed' activities in others, rather they make followup visits, refer patients to other professionals, book patients to come to the clinic at a certain time, order tests or particular treatments and so on. Each of these procedures can be represented as a (different) refinement of the ***.Embed*** operation.

Consider the act of ordering a test, or a treatment. This operation is called ***.Order***. ***.Order*** is a refinement of ***.Embed*** (and ***.OutEmbed***). An activity can be ordered by a clinician when it is of a type that cannot be run by that clinician, and has no followup types (As given in *FollowGuide*, to be explained shortly). If the type of the new activity is in HomeType, then the ***.Embed*** operation is invoked, otherwise it is ***.OutEmbed***. The operation is as follows:

ATClass6.Order$(A_p,A_b,t_c,hcp\rightarrow a_c)$

$(t_c,ProfType(hcp)) \notin RunType$

$t_c \notin Dom(FollowGuide)$

$t_c \in HomeTypes \Rightarrow$ ***ATClass5.Embed***$(A_p,A_b,t_c,hcp\rightarrow a_c)$

$t_c \notin HomeTypes \Rightarrow$ ***ATClass5.OutEmbed***$(A_p,A_b,t_c,hcp\rightarrow a_c)$

.

[xvi] In the theory itself as presented in Appendix 2, information and communications are presented after the operational refinements. The ordering in the thesis has been decided for didactic purposes, the author considering that that chosen is clearer than the alternative.

Thus a doctor might order a blood test, or Chiropodist Care, or DSN Care. A doctor cannot order a DSN Cons activity as this has potential followups (instead, that activity, if created by the doctor, would have to be booked using one of the operations ***.Book***). Although it is undoubtedly possible to refine the operations as the theory has done, the question that has to be asked is whether the refinement so presented corresponds to the name of the operation. There are occasions when a clinician 'orders' an activity (or at least would claim to have done so) - whether this occurs in those circumstances circumscribed by the preconditions of the theory is open to doubt. This is one of the least 'tested' or 'refuted' areas of the theory, so should be viewed most sceptically.

One of the refinements of the ***.Embed*** operation that is of most interest to us is ***.Followup***. The idea of the followup consultation is central to a hospital's view of medicine, especially in a predominately out-patient oriented area such as the Endocrine directorate. The underlying structure of care as it is provided for diabetics in the DEDC takes the form of an initial consultation with the doctor and followup visits every six months or so. The vast majority of diabetic care at St Thomas' is delivered in this way with the doctor having the responsibility for periodically and regularly reviewing the patient's health for as long as they are registered with the hospital. Although the paramedics do not work quite like this, they too, on seeing the patient during a consultation may decide to have them come back again in a day, a week or a month so that 'progress' can be assessed. In the theory then, certain activity types have followups. The followup to an Init Dr Cons will be a Followup Dr Cons, and the followup to a Followup Dr Cons will be another Followup Dr Cons. This property of the domain is recorded using the structure over types called FollowGuide where

T28: $FollowGuide: Types \twoheadrightarrow (Types \twoheadrightarrow Types)$.

Any member of the set *FollowGuide* will have the form $(t1,t2,t3)$, in which case the followup of an activity of type $t2$ will be an activity of type $t3$, so long as both are embedded in an activity of type $t1$. In other words, an activity has only one type of followup activity for a given parent (The case where the activity has multiple parents is dealt with in the precondition of the operation ***.Followup***). The actual followup of an activity is represented by a tree over *Activities*. Thus

T29: $Followsup: Activities \twoheadrightarrow Activities$.

If activity $a1$ *Followsup* activity $a2$, then $a1$ must be *After* $a2$. We enforce this observation through the invariant

161: $Followsup \subseteq After$.

A followup activity is never the child of more than one parent: this makes the ***.Followup*** operation clearer, but must be remembered when specialising the theory (although the invariants associated with *FollowGuide* mean that this invariant can be kept, it might make an appropriate specialisation harder to find). This invariant is defined with the predicate

162: $(Cod(Followsup) \lhd During) \in Activities \twoheadrightarrow Activities$.

Finally, there is a theorem which say that when activity $a1$ *Followsup* activity $a2$, the type of $a1$ must be allowed to followup the type of $a2$ as specified in (the codomain of) *FollowGuide*. Thus

163: $ActType \circ Followsup \circ ActType^{-1} \subseteq Cod(FollowGuide)$.

There are a number of other invariants that describe in formal terms the relation between the *Activity* tree *Followsup* and the domain of *FollowGuide*. These are too longwinded to describe here - the reader is referred to Appendix 2 for an explanation of these other invariants.

10.5.5 Boundaries

We saw briefly a representation of the boundary of the particular medical domain we were interested in at the beginning of Chapter 9 when the sets *In* and *Out* were discussed. But what do we mean by an external activity? If we claim to be constructing theories about the 'real world', how then can we construct any sort of boundary at all - after all we cannot see this boundary and surely it is not 'real' in the same sense as a patient or a clinician. The mistake is to imagine we are or could ever hope to be constructing theories of the real world: this is not the case. We are constructing a theory where behaviours of its models comply with a clinician's (or any other domain stakeholder's) view of the world. The hope throughout the work has been that there is enough in common between different domain workers that one theory can in some sense serve them all. Thus the theory is constructed so as to comply with some sort of shared perceived (or construed) reality.

Now no clinician or anyone else can hope to know of all the activities that go on in the hospital, or probably even all the activities in their own department or directorate. Equally very few clinicians will be aware only of those activities taking place in their own area: he or she will be clearest about those activities that are most immediate both in terms of responsibility and time, and 'vaguest' about those that are most distant. How a perception 'shared' between clinicians might appear is impossible to say, but the theory works according to an interpretation that has proved to lead to satisfactory models (ie the models are not of the shared perception, but are consistent with it).

This interpretation is as follows. We assume that all activities inside the boundary are known about: they are all started, and stopped, and most of them are created by the organisation of interest, that is, inside the boundary. Whenever an activity is started in a model of the theory, this is intended to represent the start of an activity in the domain. Not all activities outside the boundary are known about however, only those that the organisation has been 'told about'. Thus activities in *In* in models of the theory represent activities in the domain, whereas activities in *Out* in models of the theory represent place holders for activities that the organisation has been told about: we might say that activities in *Out* were 'ghosts' rather than the real thing. These ghost activities have many of the same properties as the internal activities - they can be requested, start and stop. However, a requested ghost activity does not have the same interpretation as a request in *In*. If an internal activity is requested in a model of the theory, this activity should be imagined as coming into being in the domain. If a ghost activity is requested in the same model, we should interpret this as a request for a service being sent out from the home organisation to some other organisation that is to perform the activity.

For example the DEDC might request a particular blood test of the pathology department, and send a sample of blood along with the request. Generally this would result in the blood test being carried out and the results returned to the day centre. Occasionally the blood sample will get separated from the test request, or both sample and request might get lost totally, or at least delayed through a problem in hospital portering. In these cases, although a test request will not be created in the pathology department, as far as the DEDC is concerned, a request has been made: this is represented by a ghost activity in a model of the theory.

In a similar way to state changes in activities, various attributes of activities should be interpreted differently depending on whether they are external or internal. For example, all 'information' that is generated and in some way recorded in an internal activity is represented in models of the theory (at least this is the intended interpretation). Information associated with ghost activities on the other hand should be interpreted as records that have been sent to the home organisation from outside the boundary. Thus in

our case of the blood test, a result might be recorded in the pathology lab, but it is not until that result has been passed to the day centre that any record would be associated with the appropriate ghost activity.

Of course, this difference in interpretation between members of *In* and members of *Out* is not defined or hinted at in the formal theory. In order to understand models of the theory however, and interpret them as representations of the 'world' that are in some way consistent with domain workers' constructions of that world, we need to be fairly clear about the distinction in meaning between the two sets.

10.5.6 Conclusion

This section has dealt with a number of subsidiary topics that are nonetheless important if we are to gain an insight into the theory and thus into the domain. Time and booking have been represented in a fairly straightforward manner, with certain types of activity susceptible to booking and certain types not. An early version of the theory represented communications between health care professionals as activities of a special type called 'Professional Dialogue'. This was abandoned along with the "activity as creating agent" paradigm. Communications between professionals is now only represented when it leads to the creation of a new activity: the content of communications now being represented in the same way as any information that can be shared around the home organisation (at least the theory does not say that it can't as interrogation of the state of the organisation is not described by the theory). The operation ***.Embed*** was refined so as to reflect more closely the sorts of operations that are observed in the domain such as referrals, booking and arranging of followups. Finally, we discussed briefly different interpretations of activities in *In* and those in *Out*. This is a symptom of the deeper problem pertaining to the exact epistemological nature of the theory. This is discussed in more depth in Section 13.3.

10.9: Conclusion to Chapter 10

Chapter 10 has examined and described enrichments rather than major change to the underlying theory presented in the previous chapter. We now have representations not only of activities and types, but also of patients, clinicians, professions, time, medical records and more. As well as adding detail to the state components of the theory, we have refined its operations to be more realistic in that they now have names and functions similar to operation types perceived in the domain.

One of the most interesting phenomena observed in this chapter is the 'paradigm shift': the change from the assumption that 'activity begets activity' to the assumption that 'clinician begets activity'. After this profound change in the theory, the conceptual structure was much simpler, although the formal presentation of that structure is probably as complex if not more so than previously. This particular problem might be solved by looking at representing the properties expressed by the theory in a clearer (though equally formal) manner. The paradigm shift helped the project move out of the rut it had got into over representation of 'professional dialogue' activities.

A summary of the 'improvements' described in the last chapter are given below:

Original Property	**Refutation / Reason for Abandonment**	**New Property**	**Discussed in Section**
Each activity has exactly one patient as its 'subject'.	Patient Education Session is attended by many patients.	Patient Education Session (and similar procedures) redefined as many parallel activities.	10.2.2

Each activity 'structure' has exactly one patient as its 'subject'.	*Activities* relating to child birth may have activities related to baby included in activities related to mother.	Problem unsolved - theory unchanged.	10.2.3
Patients may be physically present at one activity at a time.	Some Chiropody Cons activities Include a Dr Pop-in activity. The patient will be present for both of these.	A patient may be present at many activities, so long as they are all direct ancestors or descendants of one another.	10.2.4
InLoco, a 2 place relation over *Types*, dictates which types of activity can create which other types.	Relation could not represent asymmetry between different Health Care Professionals	*InLoco* changed to a 3 place relation to support asymmetry.	10.3.2
InLoco, a 3 Place relation over *Types*, dictates which child of an ancestor an activity can create.	A DSN Cons activity might be created as a result of a Dr Cons activity, but DSN Cons is not a child of an ancestor of Dr Cons.	Abandoned paradigm of activity as (activity) creating agent. New theory has a clinician as the creating agent.	10.3.3, 10.3.4, & 10.3.5
Activities run by a clinician simultaneously are in ancestral relationship.	New definition of (eg) Patient Education Session as many parallel activities.	Nothing said about clinicians and concurrent activities at all.	10.3.6
Communications represented as activity type: 'Professional Dialogue'.	Complexity of representation and 'paradigm shift' from activities to clinicians as creators of activities.	Communication represented in the same way as all information: associated with the generating activity and observed by other activities.	10.5.3

Table 3-2 Summary of properties that were refuted, the refutations, and the subsequent improvements to the theory

The final theory is expressed in a class that is the refinement or composition of eighteen others. The refinement and composition hierarchy is illustrated in the following diagram:

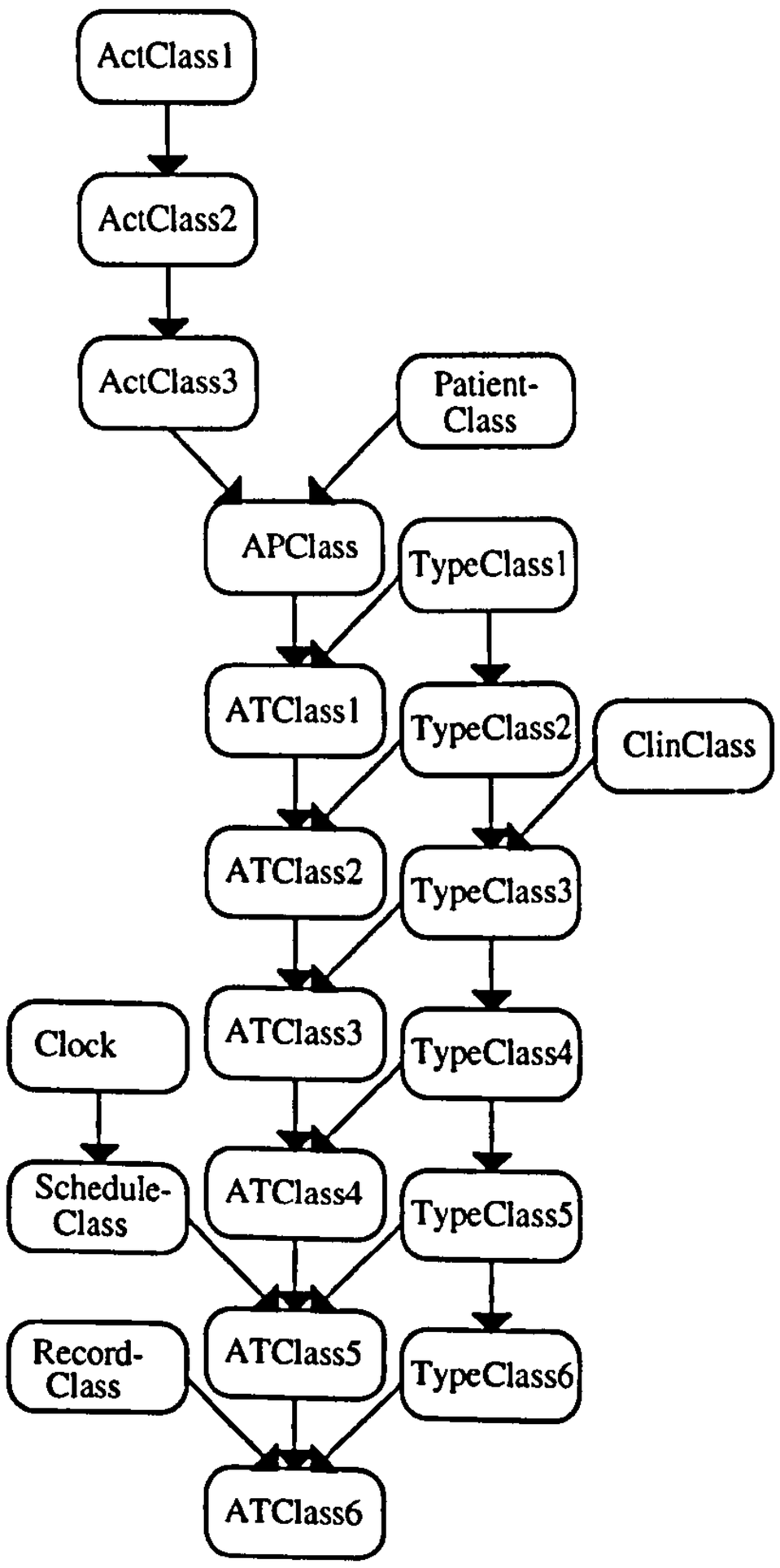


Figure 3-15: Refinement and composition diagram for all classes in the domain theory.

PatientClass and APClass introduce the set Patients and examine the interaction between *Patients* and *Activities* respectively. ClinClass introduces Health Care Professionals and professions, TypeClass3 describes some of the state components that use these concepts and *Types*, notably *RunType*. TypeClass3 is refined to TypeClass4 which defines *EmbedType*. TypeClass3 and TypeClass4 constrain the operational behaviour of the theory as explained in ATClass3 and ATClass4. Clock introduces a simple theory of time which is refined to make it more specific to the clinical domain in ScheduleClass. TypeClass5 is a refinement of TypeClass4 that defines subsets of types that are relevant to booking and scheduling (for example the set *Bookable*). The theory of time and the time related specialisation state components are composed with the operational theory as it stands to give ATClass5. Finally, a very basic theory of records and information is given in RecordClass, those specialisation state components pertinent to records are described in TypeClass6, and the whole theory brought together and presented in ATClass6.

Finally, an index is provided below for all the state components of the domain theory (sorted in alphabetic order) indicating in which class they are introduced (declared) and in which section of the thesis (if any) they are discussed. An index of classes is presented in Appendix 2 which is where the formal domain theory is presented.

StateComponent	Class Where Declared	Section Where Discussed
Access	TypeClass1	9.5.6
ActAtt	APClass1	10.2.4
ActEnd	ATClass5	Not Discussed
Activities	ActClass1	9.2.2
ActReq	ATClass5	Not Discussed
ActRun	ATClass3	10.3.6
ActSlot	ATClass5	Not Discussed
ActStart	ATClass5	Not Discussed
ActSubject	APClass1	10.2.2
ActType	ATClass1	9.4.5
After	ActClass2	9.3.2
Before	ActClass2	9.3.2
Bookable	TypeClass5	9.5.6
Complete	ActClass1	9.2.3
During	ActClass3	9.3.4
Earlier	Clock	Not Discussed
EmbedType	TypeClass4	10.3.5
FollowGuide	TypeClass6	10.5.4
Followsup	ATClass6	10.5.4
HCP	ClinClass	10.3.5
Home	TypeClass1	9.4.2
HomeTypes	TypeClass1	9.4.2
In	ActClass1	9.2.9
Includes	ActClass3	9.3.4
Later	Clock	Not Discussed
Next	Clock	Not Discussed
Now	Clock	Not Discussed
Out	ActClass1	9.2.9
OutNonCont	TypeClass7	Not Discussed
OutRefType	TypeClass4	Not Discussed
Patients	PatientClass	10.2.1
PatPres	PatientClass	10.2.1
PatReq	TypeClass1	9.5.6
Previous	Clock	Not Discussed
Proceed	ActClass1	9.2.3
ProfType	ClinClass	10.3.5
RecCont	Information	10.5.3
Records	Information	10.5.3
RecSource	ATClass7	Not Discussed

Request	ActClass1	9.2.3
RunType	TypeClass3	10.3.6
SlotClist	Scheduler	Not Discussed
SlotEnd	Scheduler	Not Discussed
Slots	Scheduler	Not Discussed
SlotStart	Scheduler	Not Discussed
TGrouper	TypeClass2	9.5.5
TypeGuide	TypeClass2	9.5.5
Types	TypeClass1	9.4.2
Unplanned	TypeClass1	9.5.6

Conclusion to Chapters 8, 9, and 10

The previous three chapters have explored the nature, content and rationale of the domain theory. In particular we have seen in many cases how refutation of early theorems has led to an improvement in the theory. We thus can now not only understand what the form of the theory is, but also why it is as it is: in this sense it has been 'justified'. It is claimed that the scientific method has guided the development of the theory, specifically the imperative to confront refutation through the construction of 'bold' theories, and the imperative to re-work the theory when one of its properties is refuted.

As a consequence of consciously embracing the scientific method the theory we have developed is rich and, if not robust, then at least more robust than it was. Whether the enrichments are in the right areas is impossible to say at this stage - the test of this lies in the ease of construction of useful IS from the domain description. This is addressed in the next chapter on the design and justification of Information System designs.

The scientific method as proposed by Popper is not without its critics [Kuhn70], [Lakatos76], [Feyer93]. Many of the problems that have been discovered in its application manifested themselves (albeit in a modest way) over the course of the project. These are discussed in more detail in Section 13.3, but we have already seen some of them such as the difficulty in pinning down the interpretation of a concept, the difficulty of knowing whether it is the general theory or its specialisation that has been refuted, and the importance of being able to 'shift paradigms'.

One major area that has not been addressed yet is the issue of organisational change. We can specialise the theory to represent an organisation at any stage in its development by invoking the *.Specialise* operation after model initialisation. We cannot thereafter change that specialisation which we would need to do if the organisation introduced even the smallest procedural change. In other words, we have enabled the rules of the game to be set up: we have not shown how to change the rules whilst the game is in progress.

There are a number of ways of dealing with this (and any implemented information system would need to have a strategy for supporting organisational change), the most basic being the recording of past structures and rules which apply to previous operational values that are no longer active (for example, completed activities). In this way historic values of the organisation do not contravene the system invariants, and future values can be made to comply with the new specialisation: this leaves those currently active operational state components (such as proceeding activities) having at times to work to the old specialisation and at others to the new one. That there are ways of representing the dynamics of organisational change is not in doubt: it was considered a problem outside the scope of this project however, and is suggested as a valid area for future development in Section 14.7.

One conclusion we ought to draw is the extreme difficulty inherent in the application of formality to the description of medicine. This is probably because the human body is such a complex entity, and the structure of medicine ought to and indeed does intimately reflect this complexity. For example, gender is generally considered to be a 'binary' phenomenon - a person is either a woman or is not a woman, in which case he is a man. This assumption is reflected in the vast majority of computer systems that impinge on this area. The assumption is generally made in medicine also, but in those areas that deal with hermaphroditism (one of which is endocrinology) and other gender 'defects' the binary concept is no longer valid, and in fact is useless even as an approximation of the truth as the interesting information lies in the manner by which the patient breaks this 'law' of gender exclusivity.

Notwithstanding the intractable nature of clinical practice and the deep problems inherent in the scientific approach, a theory of the medical domain has been developed. How are we to use this to develop Information Systems? This issue is addressed in the next chapter: Information Systems and their Interaction Theories.

Chapter 11: Information Systems and Interaction Theories

11.1 Introduction

Although an understanding of the domain may be of interest to people who work within it, we have not yet seen how the theory that has been developed is going to be of specific use to the goal of this project - the design of information system components that will help with the running of the organisation. The formal theory may very well not be of interest to the workers in the domain as their concerns might not be with those aspects of the domain that the theory has covered. In general there are major problems with the representation of informal human activity system in formal terms. The formal theory comes into its own when it is used to aid the abstract specification of 'mathematical' entities that are to be used to represent and aid the operational running of the domain. The digital computer is an example of such a mathematical entity, as are aggregates of those computers known as information systems.

If we are to construct an information system that supports the domain, what form should it take? The author has asserted earlier that useful information systems represent a perception of the world that its users can understand and 'live with' - that the user can interpret the concepts represented in the computer system into her domain. We can gain an insight into the nature and validity of the interpretation of the information system by comparing it with a reasonable understanding of world. We already have such an understanding in the form of the domain theory explored in last three chapters and in Appendix 2. By representing the behaviour of the proposed computer system in abstract terms, we can add a degree of formality to the investigation of the interaction of the information system with the domain interaction with theory. We do this by comparing and composing the information system theory, or specification, with the domain theory to give what is termed here the 'Interaction Theory'.

The way in which this process of information systems analysis will be demonstrated here is through the presentation and explanation of an abstraction of the proposed information system (or that fragment in which we are interested), and an examination of its interaction with the domain theory through the construction of the interaction theory. In the case of an information system, the most crucial aspect of this interaction is its interpretation into the domain. It is only through interpretation that the system can be considered to say or mean anything, and thus only through interpretation that it has any interaction with the user's world. The interaction theory thus records the (or rather a possible) nature of this interpretation.

It is useful to note here that the requirements process described here - the derivation of a domain theory, the proposal of an information system component, and the construction and examination of an interaction theory - is incomplete. There is a stage before the information system design that has been left out - that of selecting the area of the domain for which automated support is required. That this is a part of requirements elicitation is beyond doubt and has been investigated by many workers and reported extensively in the literature[xvii]. This issue has not been investigated in this project partly because of the extensive effort already expended in this area, and partly because in the domain where the author was based, there was not much doubt in the system procurer's mind which were the most pressing needs (though how to satisfy those needs, and how to integrate any system successfully into the functioning of the directorate was unclear to him).

[xvii] Many books that tackle systems analysis and IT strategy will address the subject of which is the most pressing sub-domain for computerised support. For an up to date insight into this and other aspects of requirements analysis, the electronic magazine: 'The Requirements Engineering Newsletter' (Back issues can be obtained via anonymous ftp from ftp-host: dse.doc.ic.ac.uk, Directory: requirements)

The next two chapters attempt to illuminate the nature of the interaction theory and its use in understanding and engineering information systems through the introduction and discussion of two examples.

In the first chapter we consider an existing information system (actually one that is all but complete but has not yet been installed). This is the departmental Clinical Record System (CRS). By considering this system in a highly abstract (and simplistic) way, we can see how we might represent an information system in terms of a mathematical theory, and how we can construct an interaction theory to assess the interpretational adequacy of the system.

Understanding the limitations of any tool makes its use much more effective, however the main purpose for the derivation of the domain theory and the construction and inspection of interaction theories is the design of new information systems. The second chapter in this logical group considers just such a new system: an integrated system which both supports the patient record and facilitates booking of clinic appointments. A hospital-wide appointment system already exists, but it is not integrated with the CRS, and does not support all the functions associated with clinic booking. By using the method explained here, we can see how we might integrate the appointments system with the CRS, and add functionality that does not yet exist (note however that the technical as opposed to the semantic aspects of this systems integration are not addressed by the thesis). We do this by first considering the appointment system in isolation, then composing it with the CRS to form a proposed integrated information system. The design of this integrated system was influenced by the existence of interaction theories: the way in which the interaction theories inspired the design is given finally.

In both these areas, an abstract specification of the information system (real or proposed) is presented and explored, followed by a development of a theory of its interaction with the domain. Formal notation and informal text will be mixed together in whatever way is considered most useful in putting across the argument, much as with the last two chapters. This chapter contains no refutative experiments however: these are appropriate to the development of the domain theory, but not to the design and investigation of information systems based thereon. Of course when designing systems, the information system design and interaction theory were developed and used iteratively: such is the nature of engineering. At presentation of the results however, for didactic purposes of clarity and comprehensibility, the information system theory and the interaction theory may be reasonably separated.

Although the purpose of this project has been to conduct analysis and design for a directorate information system, less time was spent on the derivation of these specifications than on the development of a robust domain theory. This is partly because the act of producing and refining a formal theory of a domain as complex as medical care was sufficiently difficult to occupy the majority of the time allotted to the project, and partly because it is the author's belief that once an adequate domain theory is derived (along with an understanding of the areas of the domain that require automated support) the design of information systems components, at least at an abstract level, is comparatively straightforward. In short, the development of the domain theory is the most important aspect of this method, and so most effort should be expended on it.

Those components of the directorate information system that have been specified below were chosen partly due to pressing (expressed) needs in the directorate, and partly to illustrate and explore important concepts associated with the method. The implementation of these components is outside the scope of the project, although aspects of the information systems described are being implemented now. The results

and lessons learned from the implementation of the directorate information system will be the subject of future papers.

11.2 The Clinical Record System

11.2.1 Introduction

First we will investigate the interaction between an existing information system and the domain. The information system we will investigate here is the department's Clinical Record System (CRS). As explained earlier, the departmental computer system - *Diabeta* - was being re-written at the time of the project to make it more flexible and easier to maintain. The language and system architecture were chosen so that the new system would comply with the hospital's declared 'Information Technology Strategy' [KPMG89]. This insisted on a Client-Server hardware architecture, with the server acting as the data repository running under the UNIX operating system, and the client systems being IBM-compatible personal computers running under Windows and DOS. The development environment chosen is a fourth generation language which is responsible for the creation of relational databases, and 'forms' through which the databases might be manipulated, updated and interrogated. Because of the type of development environment used, and the manner in which the new system is 'written', it was not very difficult to represent the CRS in an abstract manner using the formalism of the Schuman-Pitt notation. This is not generally the case, and for many, or even most programmes an accurate 'reverse engineering' into a formal abstract specification would be prohibitively difficult (though there are tools such as BTool and MALPAS that assist in the process).

It should be noted that the new CRS was still under development at the time of writing this thesis. The domain analysis described above, and subsequent formal and informal investigation of the interaction of the CRS with the domain (as understood through the domain theory) all influenced its design, and continue to do so. The information system described in this section is thus an abstraction of the CRS at one stage in its development.

The full and formal description (in Schuman-Pitt notation) of the CRS and its interaction with the domain is given in Appendix 4.

11.2.2 Explanation of existing components

The Clinical Record System that is used in the DEDC, known as 'Diabeta', is largely responsible for keeping diabetic records on patients (though a paper 'back-up' copy is also kept of any information stored on Diabeta which is kept with the conventional patient notes). The system is first used with regard to a particular patient when he or she arrives for their first outpatient doctor consultation, or visit. It is used to store details relating to followup visits thereafter. The original system on which the new CRS is based, only recorded information entered at a doctor consultation. This was adapted recently so that it could also support specialist nurse consultations, and more recently still, chiropodist consultations.

At a single session, the user would be presented with a patient selection screen followed, upon such a selection, by a 'demographics screen' which records essentially unchanging details pertaining to the patient such as patient address (obviously this will change, but at a much lower frequency than patient visits), GP name and address, postcode, and gender, none of which is clinical information. Thereafter the system is divided into three 'clinical pages' which can be chosen and moved between arbitrarily. The problem page lists the medical problems that the patient has had, or currently suffers from. The test results page gives a longitudinal display of the test results form the last six visits. Using this page, the

doctor or nurse can determine if there are any undesirable long term trends with weight, blood sugar level, diabetic 'control' and so on. The medication page records the drug regimens the patient has been prescribed over the last six visits. Using this and the test results pages the doctor or nurse can see what course the diabetes is taking, and what effect different drug regimens are likely to have.

The new system mimics this functionality, but has been designed with 'generality' in mind. Thus it is designed to represent many different sorts of encounters, or types of visit, and so is similar in some ways to the domain theory. Another similarity with the domain theory is the support for an inclusion relation where a record is kept of which visit is a part of which other visit. These differences with Diabeta I are partly due to the slightly different needs of the new system, and partly due to the analysis described in the thesis.

11.2.3 Description of existing components in Schuman-Pitt Notation

The Initial Operational Classes: Static Specifications

As was explained earlier, we can abstractly represent the new clinical record system using the notation we exploited to represent the domain. The theory of the CRS is built up by composing together more primitive classes, mush as we saw for the domain theory. The most basic class represents the structure of the entity *crs-Visit*. Note that all state components of the information system are denoted by the prefix '*crs-*': through the use of his prefix we will be able to more easily distinguish these sets and relations from those pertaining to the domain, something that will be useful when we construct the interaction theory.

The set *crs-Visit* should be interpreted as a data-file in a computerised database. It is through strict interpretation of the '*crs-*' state components as aspects of the information system that we can say that the formal theory we are presenting is in some way a specification of the information system. The set *crs-Visit* is of a certain data type, thus we say

$_{crs\text{-}T\,1'}$: *crs-Visit*: *Set*[*crs*-V]

where *crs*-V is the carrier set that denotes the type. This set is partitioned into *crs-Proceed* and *crs-Complete*, so we can say

$_{crs\text{-}T\,1'}$: crs-*Proceed*, *crs-Complete*: *Set*[*crs*-V]

$_{crs\text{-}I\,1}$: *crs-Proceed* $\cap$ *crs-Complete* = $\varnothing$

$_{crs\text{-}I\,2}$: *crs-Proceed* $\cup$ *crs-Complete* = *crs-Visit*

in much the same way as we did for the set *Activities* at the start of the domain theory.

The sets *crs-Proceed* and *crs-Complete* are of the same type as *crs-Visit* as indicated in their type declaration. We might imagine that each member of *crs-Proceed* is a record in an entity called *crs-Visit*, implemented as a table on a relational database system. Maybe this table contains a 'flag' field to enable a distinction to be drawn between those records that are members of *crs-Proceed* and those that are in *crs-Complete*. The details of the implementation are of no interest to us here, however: it is only important to know that the information system in some way distinguishes between the two sets.

The initial class used to define these most basic structures of the Clinical Record system is *CRSClass*1: the type declarations and invariants are given in the following schema.

***CRSClass*1**

$$\begin{array}{|l}
\hline
\textit{crs-T 1}: \textit{crs-Proceed}, \textit{crs-Complete}, \textit{crs-Visit}: \textit{Set}[\textit{crs-V}] \\
\textit{crs-I 1}: \textit{crs-Proceed} \cap \textit{crs-Complete} = \varnothing \\
\textit{crs-I 2}: \textit{crs-Proceed} \cup \textit{crs-Complete} = \textit{crs-Visit} \\
\hline
\textit{crs-Visit}' = \varnothing \\
\hline
\end{array}$$

.

This class has only two operations associated with it: ***.CreVis*** which creates a new member of *crs-Proceed* and ***.FinVis*** which takes a member of *crs-Proceed* and puts it in *crs-Complete*. We do not choose to compose this class with any domain class at this stage, as there is insufficient structure in CRSClass1 for us to gain any useful insight into its relation with the domain. We need to enrich the information system theory further if it is to be of genuine benefit to us in our attempts to understand how the CRS and the domain interact. The first such enrichment is recorded in CRSClass2. This defines a graph over the set *crs-Visit* called *crs-VisRel* (Visit Relation). In fact the graph that is defined is a tree: the relationship between the domain and codomain is functional, and is non-cyclical. We say this by putting

$\textit{crs-T 2}: \textit{crs-VisRel}: \textit{crs-Visit} \nrightarrow \textit{crs-Visit}$

and

$\textit{crs-I 4}: \textit{crs-VisRel}^{+} \cap \textit{id}[\textit{crs-Visit}] = \varnothing.$

This last invariant says that *crs-VisRel* is a directed acyclic graph.

If a visit record $v1$ that is a member of the set *crs-Proceed* is related to a visit $v2$ via the relation *crs-VisRel* (ie $(v1,v2) \in$ *crs-VisRel*), then v must also be a member of *crs-Proceed*. This property is expressed using the invariant. Furthermore, inspection of the system revealed the absence of any mechanism for moving 'parent' *crs-Visit* records to the *crs-Complete* set. These two observations are recorded in the invariant

$\textit{crs-I 3}: \textit{Cod}(\textit{crs-VisRel}) \subseteq \textit{crs-Proceed}$

which says that any parent of a visit record must be in the set *crs-Proceed*.

Interpretation of the Information System Theory: A Word of Caution

It is tempting to try and understand these type declarations and invariants by interpreting the sets and relations as concepts in the world. If this were the case we could think of a visit as an event with duration, which starts and stops. The invariants could then be interpreted as meaning that a visit cannot start unless the visit it is a part of has started, and that a visit could not stop until all visits that are part of it have stopped. This interpretation would be similar to that intended for the state components of the domain theory, *Activities* and *Includes*. This temptation must be resisted as it is wrong and misleading. The domain theory is a theory about the processes observed, or perceived, to take place in the world, or at least a particular part of the world (the domain). The state components are intended to be interpreted as aspects

of the world, and the values in models of the theory as objects or attributes of objects in the world. The information system theory is a theory about a computer system, and the sets and relations described in the theory are representations of state components of the computer system: bits and bytes, files and tables or entities and relations depending on the level of abstraction we are using to think about it.

The reason why it is important to distinguish between the domain and the information system in this way is because it is precisely the interpretation of the information system state components as aspects of the domain that we are interested in: in other words, we must be careful not to treat interpretation in a cavalier manner. It is true that when in use, the state components of the computer system are likely to be interpreted as aspects of the domain: it is our purpose here to investigate whether possible interpretations are valid, and what is missing in it. We want to do this explicitly through the investigation of the formal relation between the domain theory and the information system theory. If we are to do this, we must be sure of what the different theories are speaking: the domain theory is speaking about the 'reality' of the domain, and the information system theory is speaking about the computer system that is to be introduced (or has been introduced) into the domain. That, in turn, the computer system can be understood as talking about the domain is something that we want to investigate properly and explicitly in due course. In short, we do not want to discuss interpretations informally here: we will discuss them formally later.

The Initial Operational Classes: Dynamic Specification

As in the earlier CRS class, we have operations to create a visit record as part of the *crs-Proceed* partition, and one to move it to the *crs-Complete* partition. The CRS does not just create new visit records, it relates them to others via the *crs-VisRel* relation. There are three different sorts of visit record creation operation defined in this class: ***.CreVis***, ***.EmbInOld***, and ***.EmbInNew***. While ***.CreVis*** creates a new visit record that is not in either the domain or codomain of *crs-VisRel*, ***.EmbInOld*** and ***.EmbInNew*** each create a number of new visit records that are all related via the relation *crs-VisRel*. For example, take the operation ***CRSClass2.EmbInOld***(v_o, V_n) which returns a set of newly created visit records V_n when supplied with an existing visit record that is in the set *crs-Proceed*. Thus we have

crs-Pr 3: V_n: *Set*[*crs-V\crs-Visit*]

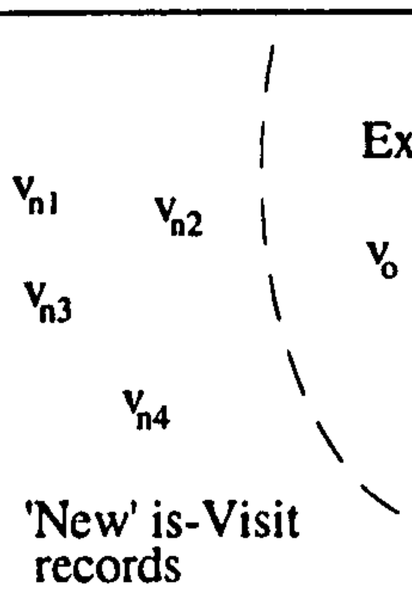
and

crs-Pr 4: v_o: *crs-Proceed.*

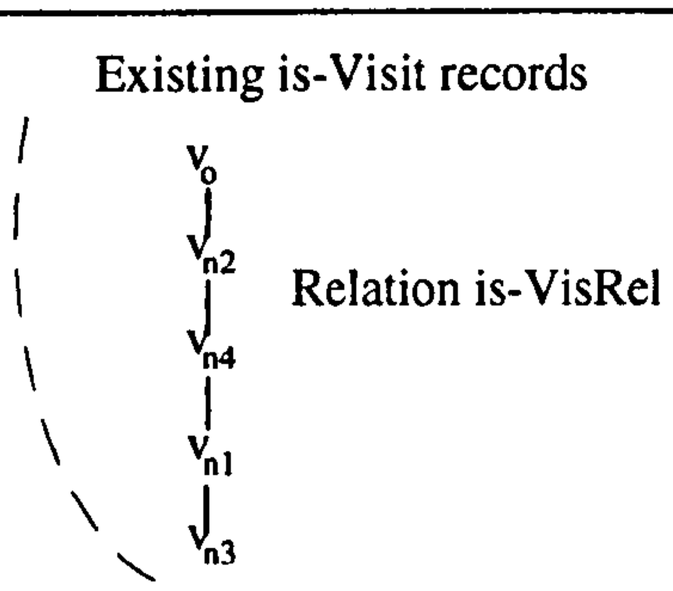
Each member of the set *Vn* is a value of the correct type (as specified by the Type declaration) but is not a member of the set (or entity or file) *crs-Visit*. After the operation all members of V_n have been created as records in the partition *crs-Proceed*: this is specified by the 'pre-condition'

crs-Pr 5: $\forall v_n : V_n \bullet$ ***CRSClass1.CreVis***(v_n)

which states that the operation ***CRSClass1.CreVis*** should be invoked for every member of the set V_n. After the operation, we want a number of new *crs-Visit* records as specified by *Vn*: we also want them to be in the relation *crs-VisRel* with each other. The intention eventually is to describe a situation where an operation to create a new *crs-Visit* record will recursively create a parent record (through *crs-VisRel*) until such time as the 'topmost' new parent visit is embedded in the existing visit record. This idea is illustrated by the following illustration: Figure 3-16.



State before operation - v_{n1}, v_{n2} etc are values of the correct type that are not yet in the set *crs-Visit*



State after operation. The vertical line represents the existence of the connected pair of records in the *crs-VisRel* relation (for example: $(v_{n3}, v_{n1}) \in$ *crs-VisRel*

Figure 3-16: (partial) States before and after invocation of the operation ***CRSClass2.EmbInOld***(v_o, $\{v_{n1}$, v_{n2}, v_{n3}, $v_{n4}\}$)

We specify this through the post-conditions:

$_{crs\text{-}Po\ 3:}$ $RestVisRel' = (\{v_o\} \cup V_n) \lhd crs\text{-}VisRel' \rhd (\{v_o\} \cup V_n)$,

$_{crs\text{-}Po\ 4.}$ $\#RestVisRel' = \#V_n$, and

$_{crs\text{-}Po\ 5:}$ $Dom(RestVisRel') = V_n$.

In these postconditions, the first predicate defines a quantity, *RestVisRel* (a Restricted version of the *crs-VisRel* relation), which is used in the other predicates. Here, *RestVisRel* is the set of pairs from *crs-VisRel* where each member of the pair is one of the new *crs-Visit* records, or the specified existing one, v_o. The other postconditions place further restrictions on this quantity. For example, the number of such pairs must be the same as the number of new *crs-Visit* records, and all the new visit records must be present in the domain of the defined subset of *crs-VisRel*. We already know that *crs-VisRel* is a function, and also a directed acyclic graph, so the only configuration possible is the one that we want - namely that after the operation, each member of *Vn* has one 'parent' from the same set through *crs-VisRel* except for the most 'senior' member which has v_o as its parent.

The operation ***CRSClass2.EmbInNew***(*V*) is similar to ***CRSClass2.EmbInOld***(v_o,V_n) except that there is no existing record to which all the members of *V* are to be related via *crs-VisRel*. The reason why this class needs three different operations to create new visit records is that each one takes different argument types: ***.CreVis*** takes no arguments and returns a new visit record; ***.EmbInOld*** takes an existing visit record and returns a set of new ones; and ***.EmbInNew*** takes no argument and returns a set of new visit records.

Again, as with the last CRS class, there is no point examining the interaction between this theory and the domain theory yet as we have not expressed enough of the structure of the information system to get any

determinism in the resulting composite theory: sometimes an operation in the domain will coincide with an operation on the information system, but we have as yet insufficient information system state components to decide which these are. In other words we are currently defining an architecture for the information system, not (yet) the services it offers.

Specialisation Classes and Their Composition with Operational Classes

The class *CRSTypeClass*1, which is not a refinement of any other class, introduces an entity called *crs-Types*, and a tree over this entity called *crs-TypeParent*. Both of these quantities are specialisation state components, and in the eventual information system theory will have similar relationship with the operational state components *crs-Visit* and *crs-VisRel* as *Types* and *TypeGuide* do to *Activities* and *Includes* in the domain theory. As *crs-Types* and *crs-TypeParent* are specialisation state components, we will not worry about operations in this class other than one called ***.Specialise***. We cannot learn anything from the operational interaction between this class and a corresponding class in the domain theory, because we have not investigated how the specialisation state components on the domain theory (which as we shall see are intended to be the interpretation of these specialisation state components in the information system theory) behave in an operational sense either.

The classes *CRSClass*3 and *CRSTypeClass*1 are composed together to give the class *CRSVTClass*1. This class relates the state components of its composite classes through the function *crs-Visit*Type and its subsequent invariant. That is

$_{crs\text{-}T7}$: $crs\text{-}VisitType: crs\text{-}Visit \rightarrow crs\text{-}Types$

$_{crs\text{-}I7}$: $crs\text{-}VisitType \circ crs\text{-}VisRel \circ crs\text{-}VisitType^{-1} \subseteq TypeParent.$

This class has only three operations - that which is used when registering a patient, that which creates a visit record, and that which finishes a visit record. The operation that creates a visit is the most complex in that it selects one of the three creation operations from *CRSClass*3 according to the type of the visit record that is to be created.

The operation ***CRSVTClass1.CreVis*** is given a type, *t*, and a patient identifier, *pid*, as its arguments, whereupon it returns a newly created visit record that has the type *t*. There may be other visit records created - whether or not they are, and what their types are is dependant on what *t* was, and what the value of *crs-VisRel* was for visits relating to *pid*. If *t* is not the domain of *crs-TypeParent* (it has no parents through the function *crs-TypeParent*) then only one new visit record is created and is given the type *t*. If *t* does have a parent through *crs-TypeParent* then either a visit record which is a potential ancestor of v_n (through *crs-TypeParent* and *crs-Visit*Type) and is in *crs-Proceed* exists, or does not. If such a potential ancestor does exist, then it is identified, and passed as one of the arguments (v_o) to ***CRSClass3.EmbInOld***(pid,v_o,V_n). If such a visit record does not exist, then the operation ***CRSClass3.EmbInNew***(pid,V) is invoked. The new visits have such types that the invariant *crs*-I7 is not contravened, and in both these latter cases the visit returned by the invoking operation ***CRSVTClass1.CreVis*** is a member of the set of new visit records. This complex operation is given in the operation schema

CRSVTClass1.CreVis($t,pid \rightarrow v_n$)

crs-Pr 21: t: *crs-Types*

crs-Pr 22. $t \notin Cod(crs\text{-}TypeParent)$

. 1

crs-Pr 23: ***CRSClass3.CreVis***($pid \rightarrow v_n$)

crs-Pr 24. $t \notin Dom(crs\text{-}TypeParent)$

---

crs-Po 13 $crs\text{-}VisitType'(v_n) = t$

. 2

crs-Pr 25: ***CRSClass3.EmbInOld***($pid,v_o \rightarrow V_n$)

crs-Pr 26: $v_o \in (im\ crs\text{-}VisPid^{-1})\ \{pid\} \cap (im\ crs\text{-}VisitType^{-1})\ (im\ crs\text{-}TypeParent^{+}\ \{t\})$

crs-Pr 27: $(im\ crs\text{-}VisitType)\ (im\ crs\text{-}VisRel^{-1+})\ \{v_o\} \cap crs\text{-}Proceed \cap (im\ crs\text{-}TypeParent^{+})\ \{t\} = \varnothing$

crs-Pr 28: $\#Vn = \#((im\ crs\text{-}TypeParent^{+})\ \{t\} \cap (im\ crs\text{-}TypeParent^{-1*})\ \{crs\text{-}VisitType(v_o)\})$

crs-Pr 29: $v_n \in V_n$

---

crs-Po 14: $(im\ crs\text{-}TypeParent^{*})\ \{t\} \setminus (im\ crs\text{-}TypeParent^{*}\ \{crs\text{-}VisitType(v_o)\} = (im\ crs\text{-}VisitType)\ V_n$

crs-Po 15: $v_n \notin Cod(V_n \lhd crs\text{-}VisRel' \rhd V_n)$

. 3

crs-Pr 30: ***CRSClass3.EmbInNew***($pid \rightarrow V$)

crs-Pr 31: $(im\ crs\text{-}VisPid^{-1})\ \{pid\} \cap (im\ crs\text{-}VisitType^{-1})\ (im\ crs\text{-}TypeParent^{+}\ \{t\}) \cap Proceed = \varnothing$

crs-Pr 32: $\#V = \#(im\ crs\text{-}TypeParent^{*})\ \{t\}$

crs-Pr 33 $v_n \in V$

---

crs-Po 16: $crs\text{-}VisitType' \circ (V \lhd crs\text{-}VisRel' \rhd V) \circ crs\text{-}VisitType'^{-1} \subseteq crs\text{-}TypeParent$

crs-Po 17: $(im\ crs\text{-}TypeParent^{*})\ \{t\} = (im\ crs\text{-}VisitType)\ V$

crs-Po 18: $v_n \notin Cod(V \lhd crs\text{-}VisRel' \rhd V)$

That this operation is so complex is interesting, and might reflect a confusion in the underlying semantics of the CRS. On the other hand, the invariants we have seen so far have not been over complex, and this operation is only of this form in order that those invariants are not contravened. The complexity is due at least in part to the need to calculate what new visits should be created when only one type is provided as an argument. As we shall see, this is caused by an artificial restriction on the sort of visits that can be explicitly created - only those which do not have children through *crs-TypeParent* - as reflected in the type declaration:

crs-Pr 21: t: *crs-Types*

and precondition

crs-Pr 22: $t \notin Cod(crs\text{-}TypeParent)$.

We now have a reasonable model of how the Clinical Record System works. Visit records are started and completed, and may, upon creation be related to other visit records via the *crs-VisRel* relation. Each visit record is assigned a type on creation which does not change. Whether a visit record can be in the relation *crs-VisRel* with another is deduced from their types and whether those types are in the relation *crs-TypeParent.* Any visit record must be associated with exactly one patient id, and two visit records related via the *crs-VisRel* relation must both be associated with the same patient id.

The class structure that has been used to describe the system should not be taken as any sort of representation of the architecture of its implementation. The notation is 'object oriented' and allows for inheritance of properties and for the refinement and composition of primitive concepts to create more advanced ones. The clinical record system as currently being implemented is based around the relational model of information storage and manipulation which is not object oriented, and does not support the composition and refinement used in this description. The class structure of the information system theory given here enables us to understand the bahaviour of the final class CRSVTClass1 relatively easily, which in turn provides an abstract representation of the behaviour of the CRS.

Of course, the implemented CRS will be much more complex than the theory we have developed, and will include data files that represent clinician identifiers, medical results, test orders, drug dosages and so on. Some of these aspects will be investigated later on in this chapter, but for the moment, we have enough structure in the theory to compose it with the domain theory to see how it interacts. This is the purpose of the next section.

11.3 An Interaction Theory for the Clinical Record System

11.3.1 Introduction

We now want to compose the information system theory with the domain theory. We will do this in two stages: CRSInteraction1 which considers the interaction between domain and information system specialisation state components, and CRSInteraction2 which does the same for the operational state components as a refinement of CRSInteraction1. Before we do this, let us consider what we need to do in order to compose the two theories.

We have seen that models of the domain theory should be interpreted as possible behaviours of (parts of) the organisation we are investigating. In a similar way, models of the information system theory should be interpreted as possible behaviours of the computer system we are investigating. As we have already seen, one of the central assumptions of the thesis is that when in use, an information system is interpreted into

the domain by the users. This interpretation is difficult to talk about: any informal discussion involving natural language statements about the domain, the information system, and the interpretation by users of the one into the other is liable to be long-winded, clumsy, obscure and prone to error. However, because we already have formal theories of the domain and the information system, we can represent (to some extent) the interpretation formally. This makes a discussion of the representational adequacy of the information system possible and clear. The theory which enables us to do this must contain both the domain theory and the information system specification: it has been called in this project the 'Interaction Theory' and it is a composition of the two earlier theories and a description of the mutual constraints each places on the other.

We represent the interpretation of the information system's state components through the introduction of an explicit 'interpretation function' which is a state component purely in the interaction theory. This is a function which relates members of state components from the information system theory to members of state components from the domain theory. The inverse of an interpretation function is a representation relation which is generally a function also (though unlikely to be total).

An example of an interpretation function is IntV which represents the interpretation of the set *crs-Visit*. This is defined in the type declaration

crs-int-T 3: $IntV$: *crs-Visit* $\rightarrow$ *Activities*.

Thus each visit record should be interpreted as an activity. We have the representation function for *Activities*:

crs-int-T 4: $RepA$: *Activities* $\nrightarrow$ *crs-Visit*

which tells us that some activities, objects in the domain, are represented as members of *crs-Visit*, records in the information system. Now if activity *a* is the interpretation for the member of *crs-Visit v*, then *v* is the representation for *a* - representation being the inverse of interpretation. In this case we have

crs-int-I 4': $IntV = RepA^{-1}$.

It should be observed that we are investigating the interpretation of an 'idealised' information system model in that we are not concerning ourselves with 'mistakes'. One of the manifestations of this assumption is the representation of the interpretation function as a total function. We will assume that we do not have two separate *crs-Visit* records for the same activity (unless the composition of the theory operations prevents this), and we will also assume that we will not have *crs-Visit* records that do not correspond to any activity at all. These are clearly not valid assumptions to take generally, but we are not concerned with the potential for information system misuse, which is always large and is difficult to analyse formally, but in its scope of validity when it is used correctly.

The manner in which the domain theory and information system theory interact can be examined through the construction of invariants that hold between state components in the two theories. All of these invariants must be predicates which incorporate an interpretation function and / or representation function. How do we discover such invariants? How do we construct the interaction theory? This question is addressed in the following sub-section.

11.3.2 How To Build an Interaction Theory.

Before we can consider a 'method' for building an interaction theory, we must consider how we intend to use it, and hence what sort of a thing it is.

We can never predict with certainty what interpretation is going to be placed on the system by its users, but what we can say is that for a given interpretation, as described by the interaction theory, the information system will provide support in identifiable areas in identifiable ways. We must make certain assumptions as to what meanings the user will impute to various information system state components: if an information system state component has similar behavioural characteristics to a domain state component, then it is probably reasonable to imagine that the latter is a fair interpretation of the first. Of course, the user might not use the same interpretation: there are many ways in which poor user interface design can obfuscate the most lucid semantic representation, but the best user interface in the world cannot redress the shortcomings of an inadequate data structure.

As a result of the impossibility of accessing the 'real' interpretation of the information system, what do we think the interaction theory is. The error lies in thinking that the theory is a description of the actual interpretation of the information system: it is not. It is rather a description of the intended and likely interpretation and use of the system, and must be created in that light. In its construction we must try to be as honest as possible, putting ourselves in the shoes of the user (there are after all countless unreasonable and obtuse interaction theories that could be constructed) both in terms of how the state components of the information system will be interpreted and how it is to be used (this amounts to a description of the 'interpretation' of operations) - that is we should be sure that the intended use is a reasonable use. The interpretation of the state component *crs-Visit* as Patients would be unreasonable, but not mathematically or logically incorrect: it would however describe an intended use where the support given to the domain by the information system was extremely poor. Clearly, experience of the use of information systems is helpful here.

The more complete the interaction theory the better. Its main purpose is to investigate the extent to which the information system can be thought of as representing the domain. Consequently we should try to construct an interaction theory that is not only reasonable, but describes the state of the domain as fully as possible - the interpretation of all information system state components should be investigated, and the system explored to see which state components of the domain can be considered to be represented.

Decisions as to the interpretation of information system state components are thus more art than engineering, but this should not discourage us. In the case of the interaction theory for the DIS1 system, we have described what seems to be a reasonable interpretation, and moulded the information system around that. We can thus say that such an information system should be, at least functionally, 'user friendly' in that its state can be interpreted in terms of primitive concepts germane to the user's understanding of her universe, and that the system behaves more or less consistently with that universe.

Having said that the construction of the interaction theory is difficult and non-trivial, the creation and exploration of a 'good' interaction theory will reveal potential errors in representation for discussion with domain stakeholders. In other words the interaction theory must be good: it does not have to be perfect. The way in which (hopefully) good interaction theories were created in the project is described below.

The finished interaction theory is a balance of interpretation functions, invariants and embedded operations that records the interpretation and use of the information system as faithfully as possible. It therefore makes sense to first of all describe the interpretations we are most sure about. For this reason we

first inspect the most fundamental and basic state components of the domain theory to see if there are any possible representatives in the information system. If there are, then simple interpretation functions linking the two are specified. Once the interpretations of the most (conceptually) fundamental components of the information system are described, basic domain operations are searched for which have an obvious and straightforward representation in the information system: these are recorded by embedding the latter operation schema in an interaction theory operation schema along with the represented operation from the domain. theory. In doing this we will have started to make assumptions about the nature of interpretation and use of the information system - we must feel happy that these assumptions are reasonable.

Once the most basic interpretations of state components and operations have been defined, derived interpretations of the more complex state components of the information system should be constructed where possible. For example, having decided that *crs-Visit* should be interpreted as *Activities* and *crs-Pid* as patients, we should look at the information system to see if there is any state component which could be thought of as an interpretation of *ActSubject*. For such an interpretation to be valid it must be consistent with the two more basic interpretations as well as the intended use of the system as defined by the bound operations. This is not as easy as it seems, and the greater the degree to which the intended use and interpretation is pinned down, the less the scope for consistent interpretation of the remaining state components. The skill in constructing the interaction theory is the balancing of the interpretations of state components and operations such that the information system says the most obvious and extensive things about the domain (ie its scope of coverage is greatest) and yet is internally consistent.

The next sub-section describes the construction and nature of one such interaction theory - that used to understand the previously described CRS.

11.3.3 The Interaction Theory I: CRSInteraction.

Let us first consider the interaction between the different sets of specialisation state components. This we do in the class CRSInteraction1. The information system theory does not describe the CRS in detail - as we have seen, there is no mention of clinicians, professions, or medical records yet. Because of the abstract nature of the information system theory, there are many state components in the domain theory that we will not use and can say nothing about at present: introducing these into the interaction theory will only serve to obscure the argument. For this reason, the interaction class is the composition of the most highly refined CRS specialisation class - *CRSTypeClass*1 - with the least refined domain specialisation class that still contains all the state components that we are interested in - *TypeClass*2.

The interpretation and representation functions are easily defined:

$_{crs\text{-}int\text{-}T\,1}$. $IntT$: $crs\text{-}Types \rightarrow Types$

$_{crs\text{-}int\text{-}T\,2}$ $RepT$: $Types \nrightarrow crs\text{-}Types$

and

$_{crs\text{-}int\text{-}I\,1}$. $IntT = RepT^{-1}$.

The representation function is declared to be partial because on inspection it transpires that there are many types of activity that are not included in the CRS. This does not tell us much as it stands - we don't know what sorts of types are or are not represented, just that the representation may be partial. We can discover more about the nature of the interaction of the specialisation state components by looking at invariants relating to subsets of the set *Types*. There are a number of these: *HomeTypes*, *Unplanned*,

Access, and *PatReq* for a start. Invariants in this area are difficult to find, as the representation of types and rules over types is much cruder in the CRS than in the domain theory. As a consequence there are no structural constraints in the former that prevent major changes in what sorts of types the information system can represent. For example, there are no visit types in the current implementation of the CRS that allow for the support of telephone calls (or any type in *Unplanned*). There are no structural properties of the various programs that have been written that prevent representation of such activity types, however, so although we might be able to say

$(im\ IntT)\ crs\text{-}Types \cap Unplanned = \varnothing$

for the particular implementation being planned in the department, this would not provide us with any deep insight into the nature of the underlying structure of the code of the CRS. The same is not true for the subset of *Types* called *HomeTypes*. There is no facility for activities conducted outside the department to be represented in the system. For this reason we can say

crs-int-I 3: $(im\ IntT)\ crs\text{-}Types \subseteq HomeTypes$

and have actually described an aspect of the intended interpretation of possible values of concepts in the information system rather than the interpretation of values that it currently has.

What other invariants should we look for? The class *TypeClass*2 defines the state components *Types*, *Unplanned*, *Access*, *PatReq*, *HomeTypes* and *TypeGuide*. The Class *CRSTypeClass*1 defines *crs-Types* and *crs-TypeParent*. We should endeavour to find invariants that link as many state components in the domain theory to ones in the information system theory as we can. So far we have seen interpretation predicates and declarations that cover *Types*, *crs-Types*, and *HomeTypes*. *crs-TypeParent* is intended to be interpreted as a set of rules that constrain which visit types can include which others - an appropriate domain state component to use to describe this would thus be *TypeGuide*. On investigation it was found that

crs-int-T 2: $IntT \circ crs\text{-}TypeParent \circ RepT \subseteq Cod(TypeGuide)$.

which tells us little more than what we would expect to be the case. Further inspection of the CRS revealed the invariant

crs-int-I 4: $(im\ IntT)\ (crs\text{-}Types \backslash Dom(crs\text{-}TypeParent)) \subseteq Access$

which says that any member of *crs-Types* that has no parents through $TypeParent^{-1}$ is a representation of a type in *Access*.

We have not learned much from the investigation of interaction at type level, partly because the invariants and rules in the domain theory are much stronger than they are in the information system. We have seen however, that *TypeGuide* is a more complex structure than its representation, *crs-TypeParent*. We would therefore expect the activity structures that can be represented and supported to be similarly cruder. That this is indeed the case is discussed in the next sub-section which describes the interaction between the operational state components of the information system and the domain.

11.3.4 The Interaction Theory II: CRSInteraction2

Having investigated the interaction between the specialisation state components from the domain and information system theories, we can attend to the operational state components. This we do in the composite class CRSInteraction2. The inheritance structure that gives rise to this class is as follows:

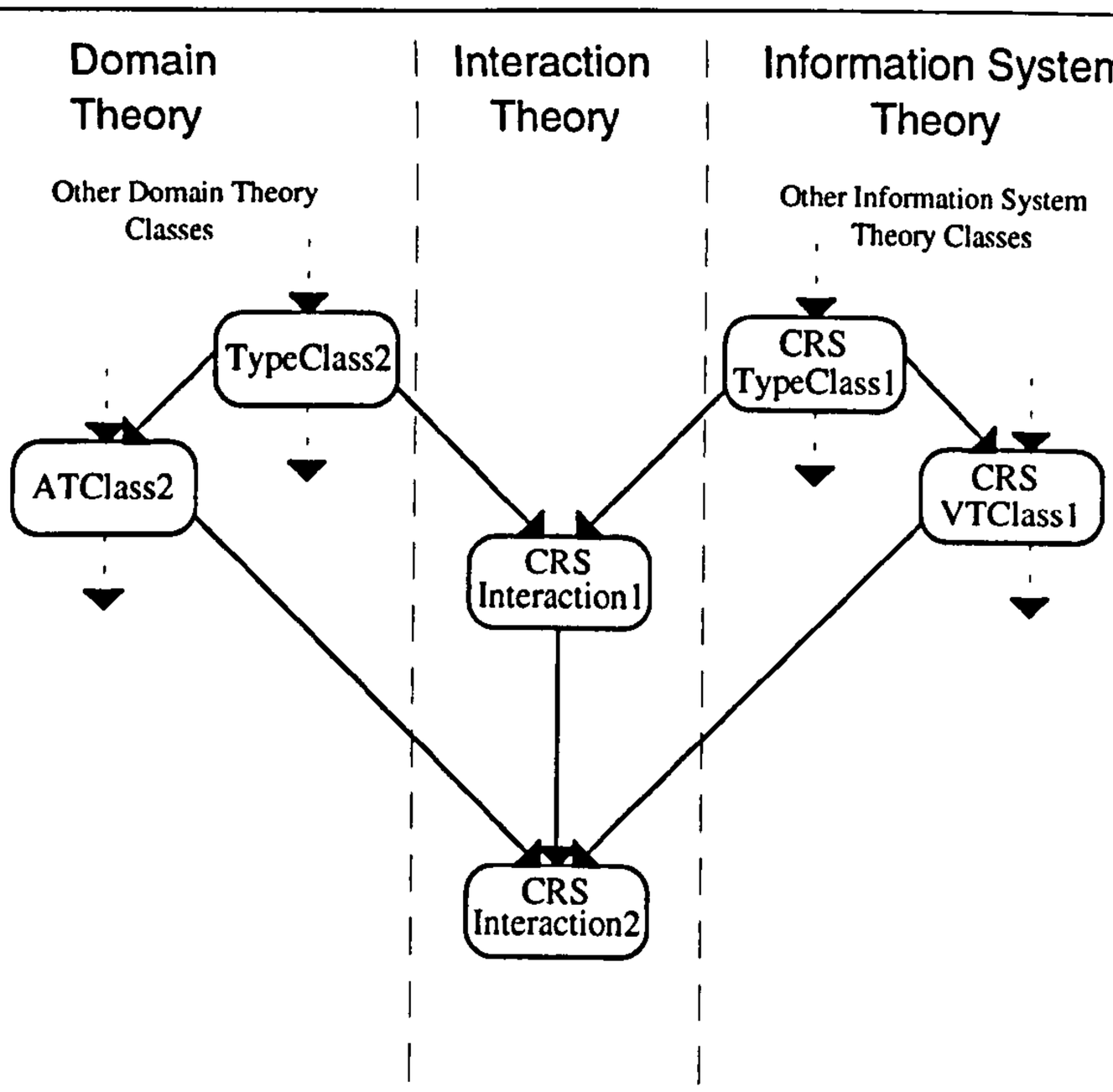


Figure 3-17: Class inheritance / composition structure for class CRSInteraction2.

As the reader will probably have guessed by now, the entity *crs-Visit* is intended to be interpreted as *Activities*. Thus we have

$_{crs\text{-}int\text{-}T3}$: $IntV$: $crs\text{-}Visit \rightarrow Activities$

$_{crs\text{-}int\text{-}T4}$: $RepA$: $Activities \twoheadrightarrow crs\text{-}Visit$

and

$_{crs\text{-}int\text{-}I5}$: $IntV = RepA^{-1}$.

There is an equally obvious interpretation for the entity *crs-Pid*, that is

$_{crs\text{-}int\text{-}T5}$: $IntP$: $crs\text{-}Pid \rightarrow Patients$

$_{crs\text{-}int\text{-}T6}$: $RepP$: $Patients \twoheadrightarrow crs\text{-}Pid$

where

$_{crs\text{-}int\text{-}I5'}$: $IntP = Rep^{-1}$.

The invariants here are more interesting than in the previous class. We first need to relate the partitions of *Activities* to those of *crs-Visit*:

crs-int-I 8: $(im\ IntV)\ crs\text{-}Complete \subseteq Complete$

or *crs-Complete* is a partial representation of *Complete*. Again, the reader has probably worked out the intended interpretation of *crs-VisRel* which is given by

crs-int-I 9. $IntV \circ crs\text{-}VisRel \circ RepA \subseteq During.$

An obvious interaction invariant is

crs-int-I 6: $Cod(IntV) \subseteq Activities \backslash Out.$

Only internal activities are represented in the CRS. Were this not the case, then either the interpretation of the *crs-Type* associated with the *crs-Visit* would be a type not in *HomeTypes* (which has been forbidden in the earlier interaction class) or we would not be able to interpret *crs-Visit*Type in the manner described above.

We might expect to see an invariant such as

$(im\ IntV)\ crs\text{-}Proceed \subseteq Proceed$

in the interaction theory. We saw in Section 11.2 that there is no facility to finish activities that have children. This means that a specialist nurse care activity might be completed, but the representation of that activity remains a record in the set *crs-Proceed*. The invariant relating *crs-Proceed* and *Proceed* would be more complex:

$(im\ IntV)\ crs\text{-}Proceed \setminus Cod(crs\text{-}VisRel) \subseteq Proceed$

in other words, all records in *crs-Proceed* that have no children are representations of activities in *Proceed*. As it happens, this invariant is excluded also: when a visit record is created, it represents a member of *Proceed*, but if that activity is suspended, it becomes a member of *Request*. There is no facility for suspending visit records implemented in the CRS, so on suspension of an activity, the visit record represents a member of *Request*. The final invariant is

crs-int-I 7. $(im\ IntV)\ crs\text{-}Proceed \setminus Cod(crs\text{-}VisRel) \subseteq Proceed \cup Request.$

The formal interaction theory also covers invariants relating *crs-VisPid* to *ActSubject*, and *crs-Visit*Type to *ActType*.

The invariants in this interaction class give us a feeling for how the information system represents aspects of the domain. We can get more of an insight into the interaction between the two if we examine the operations on the class.

We know that some requests might be represented in the CRS, that these are only those that have been suspended is recorded through the composition of operations. In particular, the operations in the composite class that create activities do not invoke any CRS operation. Thus we have the two operations

CRSInteraction2.InCreate$(A_b, p_n, t_n \rightarrow a_n)$

crs-int-Pr 1: ***ATClass2. InCreate***$(A_b, p_n, t_n \rightarrow a_n)$

, and

CRSInteraction2.Embed$(A_p,A_b,t_c \rightarrow a_c)$

crs-int-Pr 2: ***ATClass2.Embed***$(A_p,A_b,t_c \rightarrow a_c)$

which create an activity in the domain, but do not change the state of the information system. The first activity that does that is ***CRSInteraction2.Start***(a) which is given the definition

CRSInteraction2.Start(a)

crs-int-Pr 4: ***ATClass2.Start***(a)

. 1 crs-int-Pr 5: $ActType(a) \in Dom(RepT)$

crs-int-Pr 6: $IntT \circ ((Cod(\{RepT(ActType(a))\} \lhd crs\text{-}TypeParent^*)) \lhd crs\text{-}TypeParent) \circ RepT \subseteq ActType \circ ((Cod(\{a\} \lhd During^*)) \lhd During) \circ ActType^{-1}$

crs-int-Pr 7: ***CRSVTClass1.CreVis***$(RepT(ActType(a)),RepP(ActSubject(a)) \rightarrow v_n)$

crs-int-Po 1: $(v_n,a) \in IntV'$

.

This operation schema says that a *.Start* operation in the interaction theory is synonymous with a .Start operation in the domain theory. However, the *.Start* operation in the information system theory is invoked only if a number of preconditions are satisfied. Firstly the type of activity a must be represented on the information system. Secondly, the activity structure of which a is a part must be such that the operation can be invoked without leading to an inconsistency between the new activity structure and the new structure of *crs-Visit* records. The complexity of the second pre-condition reflects the representation of a graph in the domain theory (*During*) as a tree in the information system theory (*crs-VisRel*). The start of an activity can only be recorded by the information system if the parents of that activity contain one that is of a type whose representation is the allowed 'TypeParent' of the representation of the activity's type: and the parents of that second (parent) activity contain one that is of a type whose representation is an allowed 'TypeParent' of the representation of the second activity's type, and so on.

Once we have limited the applicability of the operation in this way, the postcondition is straightforward, merely identifying the most 'junior' of the newly created *crs-Visit* records as representing the commenced activity. This is only possible because of the nature of the postconditions for the relevant cases in the *.CreVis* operation from the information system specification. Each new *crs*-visit record has one parent that is of the *crs*-type given in *crs-TypeParent*. As long as there is a potential activity for the *crs*-visit record to be matched to through *IntV* (and the pre-condition tells us that there will be), then the invariants interpreting *crs-VisRel* and *crs-VisitType* ensure that there is only one valid state following this operation. The succinctness of the representation of the postcondition in cases such as this is one of the advantages of the idea of minimum change for operations (discussed earlier in Section 6.2.9)

Having invoked the *.Start* operation, assuming the precondition are met, we may have created a record in *crs-Visit*. If we then invoke the operation *.Suspend*(*a*) which is defined

CRSInteraction2.Suspend(*a*)

crs-int-Pr 8: ***ATClass2.Suspend***(*a*)

we see that a is now a member of *Request* while its representation (*RepA*(*a*)) is a record in *crs-Proceed*. This operation is the cause of the complex interpretation invariant *crs*-int-I7 seen above.

There is one further area where the interaction theory is useful, but where the mode of use is more subtle. This concerns operations which are permitted in the information system but whose interpretations are forbidden in the domain. We said above that we were not concerned with misuse of the information system. This is by and large true, but the smaller the scope for misuse we can present to the user, the more robust the system. We can find these operations by looking for behaviours in the information system alone that cannot be observed when it is interacting correctly with the domain. An example of this is the possibility of multiple *crs*-visit records in the *crs-Proceed* set with the same *crs*-type and with the same parent. To prevent this from occurring there would need to be an invariant of the form

$$\#crs\text{-}VisitType \circ (crs\text{-}VisRel^{-1} \rhd crs\text{-}Proceed) = \#(crs\text{-}VisRel^{-1} \rhd crs\text{-}Proceed)$$

in the specification of the CRS. This means that two clinicians could claim to be seeing the same patient at the same time in two separate activities of the same type. Assuming that the domain theory is correct the existence of such a state will be erroneous. Interpretational problems of this sort are discussed in greater detail below in Section 13.3.

11.4 The Interaction Theory: Conclusion

By looking at the invariants and the operations in the interaction theory, we can get an idea of the scope of the information system, the operations (in the domain) that it supports and the reasonableness of its interpretation. The formality of the interaction theory helps in all these areas. We have a better idea now of which activities are represented in the CRS, and at what stage they become so - as indicated by the invariants and the four operations described above.

By inspecting the interaction invariants we can get a feel for how reasonable the interpretation of the information system is. For example the invariant described earlier linking *crs-Proceed* to the domain:

crs-int-I7: $(im\ IntV)\ crs\text{-}Proceed \setminus Cod(crs\text{-}VisRel) \subseteq Proceed \cup Request$

shows that the interpretation of *crs-Proceed* is far from straightforward. The desire to create a simpler invariant and finding it wrong led to the discovery of a discrepancy between the CRS and the domain theory - namely that there is no facility for completing activities that have children.

Inspection of the operational preconditions tells us the scope of applicability of the information system operation compared with the domain operation it is supposed to support. For example, in the case of the start of an activity in the interaction theory, we saw that a complex precondition had to be satisfied before a similar operation could (validly) be invoked in the information system. This tells us that not only must the activities be of a given type, but the types of the ancestors of that activity must be of a certain subtle structure. In practice this means that if the record of a blood test can be created as part of the record of a doctor consultation, one cannot be created as part of the record of a nurses consultation.

Finally, if we find classes of behaviours that are allowed by the information system specification but forbidden when composed with the domain theory to form the interaction theory, then we have observed an increased potential for error. For example the information system permits the representation of an illegal state where multiple proceeding activities of the same type share a parent.

If a discrepancy is found between the information system and domain theories, we do not have to assume there is a mistake or deficiency in the information system for two different sorts of reasons.

Firstly the discrepancy might not lie in the information system. The theory of the information system might be incorrect. The domain theory might itself be wrong (and we will see later an example of a theorem of the domain theory that was discovered to be refuted only after and because of the development of the interaction theory). The interpretation of the information system theory into the 'real' domain might be more reasonable than the interpretation of the domain theory into the domain. The interaction theory invariants might themselves be inaccurate, that is the interaction theory may have been poorly constructed - this is a special danger here as the various theory constructions have proceeded without using formal proof, only informal reasoning over formal structures.

Secondly, even if the discrepancy is a 'genuine' one where the fault lies with the information system, it might be no concern to the system users that the scope of domain representation is limited or inaccurate. For example, it has not been a great cause for concern hitherto that the CRS has not represented requests or bookings. Similarly the lack of a function which enables visits to be interrupted has not given rise to much complaint. In short, we do not have to insist that the information system is a model of the domain (or shared perception thereof): the domain theory is not a statement of system requirements. The availability of the interaction theory enables debate about future system enhancements to be held in a more informed manner, however. If problems do arise in the enlargement of the CRS (and they have) then the interaction theory might be able to point to the cause of the difficulties, and through its inspection the system developers might gain insight into how the problems should be tackled. Areas where the domain theory and interaction theory have been of practical use already are discussed in Section 14.6.

It should be noted that the conceptual structure of the re-designed CRS was carried out in parallel with the development of the domain theory, and the ideas from this work influenced the new system. If this had not been the case, the construction of an interaction theory might have been significantly more difficult: for example, it would be much more difficult to compose a theory of the existing operational system (written in APL) with the domain theory.

In the next chapter we will investigate further the uses of an interaction system. Specifically we will consider how it influenced the creation of a specification for a system that integrated the CRS with a hospital system: the Outpatient Appointment System.

Chapter 12: DIS1 - An Integrated Appointment and Clinical Record System

12.1 Introduction

In the last chapter we saw the construction of an interaction theory to help examine an existing system. This helped us understand the functions of the system as well as some of its potential interpretational shortcomings. However, we can also use such a theory to help us design new systems. This chapter explores how an interaction theory helped in the creation of one such specification, for an integrated record system and outpatient appointment system.

The domain analysis revealed several areas that could potentially be supported through the use of computer technology. These included more realistic representation of patient activity and storage of notes generated therefrom, the setting up of a clinical electronic mail system that supported internal referral of patients, a more efficient order communications system that more accurately reflected the way in which tests are requested and used, and an appointments system that integrated effectively with the running of the clinic and with other systems in the hospital. From this list the integration of the record system with the appointment system was given the highest priority. More important still was the construction of a contract management system: this is not the representation of a current aspect of the domain however, and so is tangential to the concern of this thesis. It is the representation of a hypothetical domain that would exist were the directorate to efficiently secure and maintain contracts. Such a hypothetical domain is described in Appendix 6. This chapter concerns the creation of a system that integrated the CRS with the hospital's outpatient appointment system.

An appointments system is scheduled to be implemented over the next few months in the hospital. This will happen in two stages with the outpatient system following the inpatient system. The domain analysis has concentrated mainly on the outpatient side, and this is where support is needed most urgently for the directorate (inpatient appointments are currently dealt with centrally).

This chapter is presented in three sections. The first describes the salient features of the proposed outpatient appointment system through the use of a 'reverse engineered' specification. The second describes the intended integrated system. The third explains how the use of an interaction theory influenced the design of the specification of this integrated system.

12.2 A Specification of the Outpatient Appointment System (OPAS)

There is an informal implied domain theory represented by the appointment system. In order to understand some of the constructions below it would help if we first considered this as it behaves in an unusual way.

The central concept is the slot which has most of its attributes pre-defined: the clinic it belongs to, the 'stream' in that clinic (several streams will run in the clinic on the same day), its scheduled start and stop time, the day it is to take place on and so on. The slots are defined several months in advance for a particular clinic. A booking is made when a patient is assigned to a slot - the identity of the slot a patient is associated with thus determines when the patient should arrive and what clinic they should attend.

The structure of the specification reflects this underlying conceptual structure. The first class, OPASClock (OPAS is an acronym standing for Out-Patient Appointment System), describes the representation of time that is implemented in the appointment system. This is inherited by OPASSlots, the class which describes the creation of slots. The 'clinical' attributes of slots are described in OPASClinics which composes the

slot definitions with a number of configuration, or specialisation, state components from the class OPASConfig. The assignment of patients to slots is dealt with by a refinement of this class called OPASAppt.

This representation of time by OPAS is based around a sequence of 'day' records, and a sequence of records corresponding to times of day. The specification reflects this through the definition of two ordered graphs. The first graph is over the infinite set *as-Days* and is called *as-Precedes_D*, the second is over the finite set *as*-TofD and is called *as-Precedes_T* (note that all state components of the DIS1 system are denoted by the prefix '*dis-*'). The type declarations for these graphs are as follows:

as-T 3': *as-Precedes_T*: *as*-TofD ⇸ *as*-TofD

and

as-T 4': *as-Precedes_D*: *as*-Days → *as*-Days

These two graphs are constrained to be directed and acyclic through the use of appropriate invariants.

We insist on the total ordering of these two graphs, or trees, by saying that their inverses are also trees. Thus we say:

crs-T 3': *as-Follows_T*: *as*-TofD ⇸ *as*-TofD

where

as-I 1': $as\text{-}Precedes\_T^{-1} = as\text{-}Follows\_T$

and similarly with *as-Precedes_D*. In fact, both the functions *as-Precedes_D* and its inverse are total. This, together with the fact that both structures are also directed acyclic graphs (as specified in invariant as-I 2), the set *as*-Days must be infinite. Of course the implementation does not represent the infinite set explicitly, as to do so would require infinite storage. The set would probably be generated algorithmically, but this is of no concern to us here. There must be a first and a final time record for each day record so we say

as-T 1': *as-FirstT*, *as-LastT*: *as*-TofD.

This class also introduces a function which returns a natural number when applied to a pair from the set *as*-TofD. Thus

as-T 5: *as-Duration*: *as*-TofD x *as*-TofD ⇸ N.

This represents the duration of the interval between the times represented by two *as*-TofD records. The mutual constraints between these different state components are expressed through a number of invariants that we need not go into here, the reader being referred to Appendix 5 for the full specification.

It might be remarked that the use of sequences to express the orderings of the various sets described would be more elegant and clear. This is probably true, but the use of more sophisticated primitive structures has been avoided throughout the various theory presentations derived from the project - state components are sets, relations, or graphs and we see no bags or sequences. The use of such simple basic structures means that we only need a minimum of set-theoretic operators to construct the desired description. For this reason we do not need to bother with the definition of such concepts as bag union,

sequence concatenation, squashing and the like. Whether the resulting description is actually clearer is questionable, and a reworking of the various theories using more sophisticated primitive structures might well be profitable.

The class OPASSlots introduces a set called *as-Slots* and defines attributes of members of this set. Remember that in the informal domain around on which this system is based, each slot is associated with a stream, a day, a start time, an end time and a duration. Having defined the representation of time that is used in the appointment system, we can use the concepts introduced in the earlier class to express the way in which the OPAS represents these relations and their properties. For example we say that

as-T 10: *as-SlotStart*, *as-SlotEnd*: *as-Slots* → *as*-TofD

and

as-T 11: *as-SlotLength*: *as-Slots* → N

where the value of *as-SlotLength* is given by the invariant

as-I 11: *as-SlotLength* = *as-Duration* ° (*as-SlotStart* ◊ *as-SlotEnd*).

In other words, that the 'duration' of a particular slot record is the same as the duration returned by the *as-Duration* function applied to the start time record associated with the slot record and the associated end time record.

The stream record provides a method for aggregating members of *as-Slots*. For a given stream record no two as-slots can be 'overlapping': if the start time record of an as-slots is 'before' (as defined by *as-Precedes_T*) the start time record of another, then so must be the end time. The *as-Slots* records need not be contiguous however: this reflects the situation in the domain where there are gaps in the clinic sessions for lunch breaks, tea breaks and so on.

Operations on the class support the creation and deletion of stream records, and the creation of as-slots and their assignment to as-streams. This operation is intended to be used once every few months to create all the slot records needed to support clinic bookings in the immediate future.

OPASConfig describes the specialisation state components that are used to configure the system to a particular aspect of medical care. Again, referring to the implicit domain theory we see that the set *as-CTypes* represents the type of the clinic to be supported while *as-ApptMode* represents the set of allowable modes of session. Two slots of the same mode will have the same duration: the introduction of different modes allows the support of situations where slots of different lengths are permitted in a single clinic stream (such as would be the case in an outpatient clinic that saw both new and followup patients).

Members of the set *as-CTypes* can be associated with members of *as-ApptMode* through the relation *as-CTypeModes* - for example a diabetic clinic might support both new and followup patients while an inoculation clinic makes no such distinction. Each pair in *as-CTypeModes* is associated with a number. This represents the duration of slots associated the clinic type and appointment mode represented by that pair. This is done through the function *as-CModeLength*.

This property is expressed in the next class - OPASClinics. This class introduces a number of direct and indirect attributes of slots. For example each slot is given a mode directly thus:

as-I 14: $as\text{-}SlotMode: as\text{-}Slots \rightarrow as\text{-}ApptMode$

whereas the member of *as-CTypes* associated with a slot can be derived from the construction

$as\text{-}ClinicType \circ as\text{-}StreamClinic \circ as\text{-}SlotStream$

where as-ClinicType is the member of *as-CTypes* associated with a particular clinic record, as-StreamClinic the clinic record associated with a particular stream record, and *as-SlotStream* the stream record associated with a particular slot record. The invariant that determines the 'length' of as-slots is thus

as-I 15: $((as\text{-}ClinicType \circ as\text{-}StreamClinic \circ as\text{-}SlotStream) \lozenge as\text{-}SlotMode) \circ as\text{-}SlotLength^{-1} \subseteq as\text{-}CModeLength^{-1}$.

The final class in this 'reverse engineering' of the appointment system is called OPASAppt. This associates slot records with patient identifiers via the partial function *as-Appointments*. Additionally a slot that is associated with a patient through the *as-Appointments* function may also be associated with an arrival time record, a start time record and an end time record. The relevant invariants here are

as-I 18: $Dom(as\text{-}ArrivalTime) \subseteq Dom(as\text{-}Appointments)$,

as-I 19: $Dom(as\text{-}StartTime) \subseteq Dom(as\text{-}ArrivalTime)$,

and

as-I 20: $Dom(as\text{-}EndTime) \subseteq Dom(as\text{-}StartTime)$.

The operations that are pertinent to this class are those that assign patient ids to slot records, update those assignments and delete them. There is no explicit registration operation - the registration being dealt with at the same time as the initial booking. The booking of a patient is easily represented using the existing state components. The preconditions for the operation ***OPASAppt.BookKnownP***(pid,sl) are simply

as-Pr 29: $pid: as\text{-}Pid$,

as-Pr 30: $sl: as\text{-}Slots \backslash Dom(as\text{-}Appointment)$

which ensures that the slot record is not currently in use by any other appointment record, and

as-Pr 31: $as\text{-}SlotDay(sl) \in (im\ Precedes\_D^{+})\ \{as\text{-}NowD\}$

which ensures that the slot record that is being assigned is one that is assigned to the record of some future day. The operation ***OPASAppt.PatArrive***(sl,τ) allocates an arrival time record to the slot record, ***OPASAppt.ApptStart***(*sl*) a start time record, and ***OPASAppt.ApptStart***(*sl*) an end time record. The first of these operations has the time record provided explicitly as an argument whereas the (records of) times that are used for the latter two operations are always *as-NowT*. It is envisaged that the ***.PatArrive*** operation might be invoked after the patient has arrived at the clinic, possibly during the consultation, so the assumption that the appropriate time is always *as-NowT* is in this case incorrect.

This then is a brief description of the behaviour of the Outpatient Appointment System that is to be implemented at St Thomas' over the course of the next year. This theory has much in common with the domain theory in that it is a formal description of a system that already exists and can thus be refuted.

Refutation of the theory is significantly easier in this case as properties can be tested by 'trying them out' on the information system in question. Before a system such as that proposed in the next section could be constructed, the validity of the theory, or specification would have to be carefully checked against the actual appointment system.

A diagram indicating the class structure of the OPAS theory is presented below:

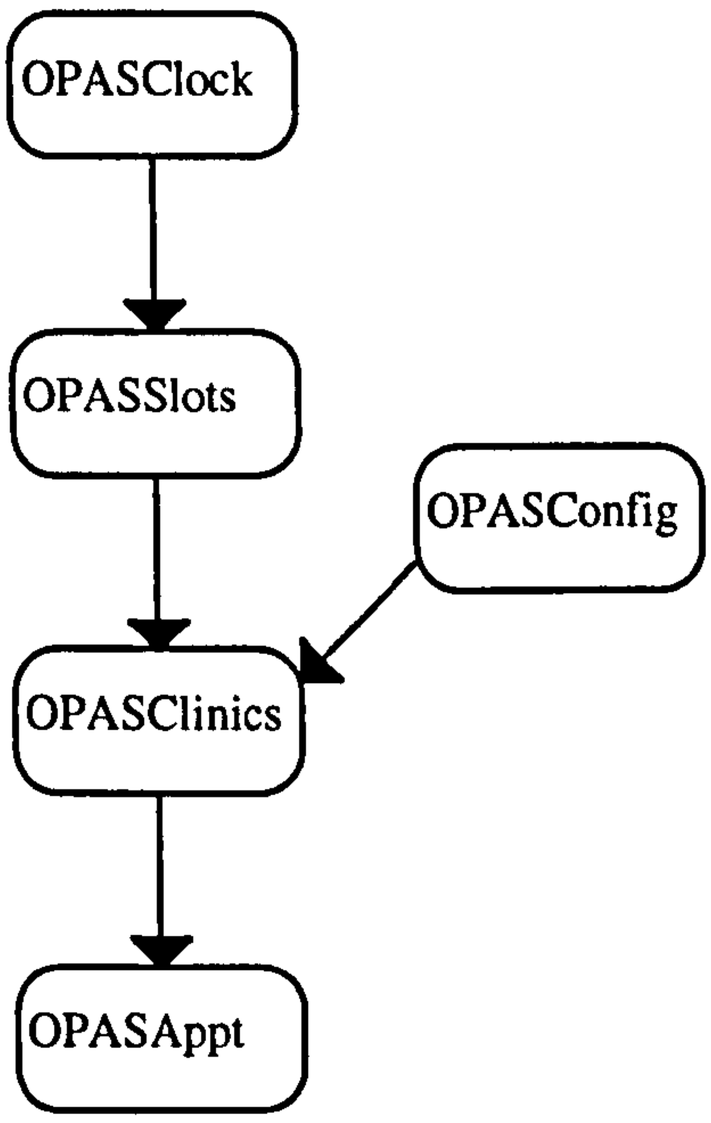


Figure 3-18: The class structure used in the theory of the Outpatient Appointment System (OPAS) described above.

12.3 A Composition of OPAS with CRS - A Directorate Information System (DIS1)

The purpose of carrying out this 'reverse engineering' to arrive at a specification of the appointment system is to enable us to integrate it with the existing clinical record system. We want this integration to support relevant behaviours of the directorate - this is investigated through the interaction theory. The composite system incorporates both the CRS and the OPAS, and introduces additional state components and constraints over those. The composite system is a fragment of the eventual integrated directorate information system, and has thus been called DIS1.

This section describes the specification, or theory, of DIS1 as a composition of OPAS and CRS. The next section investigates the way in which it was constructed with the help of an interaction theory. The sequentiality of presentation is intended to aid the reader by introducing new concepts slowly - it does not reflect the way in which the work was carried out however. The existence of a domain model influenced the design implicitly and explicitly. Implicit influence came through an understanding of the domain theory and a feel for the sort of interaction theory that was desirable. Explicit influence came directly from the formal interaction theory, developed after the initial DIS specification, but causing subsequent designs to be changed. The appropriate place for discussions of the nature of the interaction of the information system with the domain is precisely during the presentation of the interaction theory - this is after all what the interaction theory is for. It should nevertheless be remembered that the design of the DIS did not occur independently of the interaction theory - rather the two were developed in parallel.

There are two classes used to specify the information system DIS1. These are DIS1Types that describes the specialisation state components, and DIS1Class1 that describes its operational behaviour. DIS1Types composes the specialisation classes from each of the OPAS and CRS class groups - namely OPASConfig and CRSTypeClass. DIS1Class1 composes DIS1Types with the most refined operational classes from the OPAS and CRS class groups - namely OPASAppt and CRSClass4.

DIS1Types introduces three subsets of the Set *crs-Types*. These are *dis-Bookable*, *dis-Accessible* and *dis-Unplanned* (note that all state components of the DIS1 system are denoted by the prefix '*dis-*'). These will be used in *DISClass*1 to segregate the various operations that are responsible for the creation of new activity records. Only those activity records that are to be associated with a type record in *dis-Bookable* may be created via the ***.Book*** operation, only those that are to be associated with a type record in *dis-Unplanned* may be created via the ***.SuddenStart*** operation, and only those that are to be associated with a type record in *dis-Accessible* may be created via the ***.Create*** operation.

The partial function *dis-TypeLink* relates the set *crs-Types* with the set *as-CTypeModes* which plays a fairly similar function in the appointment system. Each member of *crs-Types* can be mapped onto a member of *as-CTypeModes* - the reverse is not true as there may be slots that are intended to run in the same clinic and take the same amount of time, but are nevertheless different sorts of medical activities. As we shall see, all the visit records (actually now activity records) that are of types in the domain of this function might have been created through the ***.Book*** operation: consequently we need the invariant

dis-T 2: $Dom(dis\text{-}TypeLink) = dis\text{-}Bookable$

which ensures that any activity record that can be associated with a slot is of a type that can be created via the ***.Book*** operation.

Note that all the state components are subsets of and structures over existing state components (we have not had to use any carrier sets). The scope of the system is the same at the specialisation level as the scopes of the CRS and the OPAS. This is not true of the operational integration class DIS1Class1.

DIS1Class1 introduces *dis-Activities*, *dis-Request* and *dis-Clist*, all of which are defined in terms of a carrier set. For example we have

dis-T 3: $dis\text{-}Activities, dis\text{-}Request: Set[crs\text{-}V]$.

The set *crs*-V has been seen before in the class CRSClass1 as the carrier set for *crs-Visit*. What this means is that the set *dis-Activities* is of the same data type as the set *crs-Visit*, but not necessarily containing the same records. That the sets *crs-Visit* and *dis-Activities* are of the same data type means that such expressions as $dis\text{-}Request \cap crs\text{-}Visit$ and $dis\text{-}Activities \setminus crs\text{-}Complete$ are valid, that is they have are well formed within the particular version of set theory that we are using. In fact this class introduces the invariant

dis-I 5: $dis\text{-}Request = dis\text{-}Activities \setminus crs\text{-}Visit$

which says that *dis-Activities* is partitioned into *dis-Request* and *crs-Visit* (which is itself partitioned into *crs-Proceed* and *crs-Complete*).

Any activity records that are not present in *crs-Visit* will not have types via *crs-Visit*Type or associated patient identifiers via *crs-VisPid*. The specification of the functions *dis-ActSubject* and *dis-ActType* redresses this shortcoming thus:

dis-T 6': *dis-ActSubject*: *dis-Activities* → *crs-Pid*

and

dis-T 6': *dis-ActType*: *dis-Activities* → *crs-Types*

(the next section illustrates further the value of such an introduction). These functions are supersets of corresponding functions found in the specification of the CRS - in fact if all records in the set *dis-Requests* are removed from the domain of *dis-ActType* (or *dis-ActSubject*) we are left precisely with *crs-Visit*Type (or *crs-VisPid*):

dis-Request ◁ *dis-ActType* = *crs-VisitType*

and

dis-Request ◁ *dis-ActSubject* = *crs-VisPid*

(these expressions can be derived from dis-T 6, dis-I 5, and dis-I 6).

Another state component introduced that is not a derivation of an existing one is the clinic list:

dis-T 5: *Clist*: *Set*[*dis*-Cl].

This set allows stream records to be aggregated. A clinic list is represented as a sequence of stream records associated with different as-days (remember that each stream record is associated with a particular day record via *as-StreamDay*, and that all slot records associated with that stream record are also associated with that stream's day). The way we achieve this aggregation is through the function *dis-StreamClist*.

dis-T 8: *dis-StreamClist*: *as-Streams* → *dis-Clist*

and the invariant

dis-I 14: (*dis-StreamClist* ◊ *as-StreamDay*)$^{-1}$ ∈ (*dis-Clist* x *as-Day* ⇸ *as-Stream*)

which says that for any Clist record, every stream record associated with it is assigned a different day record through as-StreamDay.

If we are to integrate the OPAS and the CRS, we need to specify some state components and invariants that link and mutually constrain the two theories. Without such mutual constraint the composite specification would consist of two orthogonal parts that were totally un-integrated. Examples of state components linking the two concepts are *dis-ActSlot* which indicates which activity record a particular slot record might apply to:

dis-I 7': *dis-ActSlot*: *dis-Activities* ⇸ *as-Slots*,

and *dis-Pidas* which ties the patient register in the CRS to that in the OPAS:

dis-I 17: $dis\text{-}Pidas: crs\text{-}Pid \twoheadrightarrow as\text{-}Pid.$

In the case of *dis-ActSlot*, we have the invariant

dis-I 12: $dis\text{-}ActSlot^{-1} \in Dom(as\text{-}Appointments) \rightarrow dis\text{-}Activities$

which insists that every slot that is associated with a patient id through *as-Appointments* has an activity record linked to it via *dis-ActSlot*. The existence of these state components enable us to place powerful mutual constraints on the two systems as a result of the composition. For example, we cannot now invoke the operation which records a booking for a patient without having assigned that patient id to an activity record. Furthermore, the invariant

dis-I 18: $(im\ dis\text{-}ActType)\ Dom(dis\text{-}ActSlot) \subseteq dis\text{-}Bookable$

specifies that each of these activity records must be of a type in *dis-Bookable*. The possible states of one system are thus severely constrained by the states of another. This is precisely what we mean when we talk about system integration. If the CRS and OPAS were not at all integrated, the composite class would be empty of mutual constraints, and the 'state space' of the new system would be the product of the state spaces of those systems that comprise it (multiplied by the state space implied by any state components introduced in the new class). As it is the state space is constrained by every invariant - each of which describes a manner in which one system interferes with another. A similar argument could be made for the space of possible behaviour traces before and after mutual constraints have been introduced.

In addition to the types of invariants described above, we can introduce some that place constraints on state variables that come entirely from one of the component subsystems. These are introduced to constrain operations that were not described in the components classes or simply to add semantics that were left out of the earlier system design. An example of such an invariant is

dis-I 21: $\#(crs\text{-}Proceed \lhd crs\text{-}VisPid) = \#(crs\text{-}Proceed \lhd crs\text{-}VisPid) \circ crs\text{-}VisitType^{-1}$

which constrains values of state components all of which are specified in the CRS system. The reason for the introduction of this predicate is explained in the next section.

If we investigate the operations supported by this class, we see that some totally new, some are refinements of a single operation from an earlier class, and others are compositions of several operations from the component classes (with additional pre-conditions).

An example of an operation that is totally new is that which is responsible for creating new activity records and placing them in the set *dis-Request*. This is

$DIS1Class1.Generate(t_c, pid \rightarrow a_c)$

taking as its arguments a patient id and an element of the set *crs-Types*, and returning a new activity record as a member of *dis-Request*. Because the relation *crs-TypeParent* is a tree, and because of the invariant that says that an activity record cannot have two 'child' records in *dis-Request* (through *dis-VisRel*) that are of the same type, we know what types of activity record must exist and be related to the new record through *dis-VisRel*, although we may not know the 'names' of these activity records. These activity records that are the 'ancestors' of the new record through *dis-VisRel* may or may not exist before the operation. Some of the preconditions thus set up a set of activity records that do not yet exist and are to be in the relation $dis\text{-}VisRel^{+}$ with the new record after the operation (this may of course be the null

set). For the operation to be invoked, there must be at least one activity record that exists, is not complete, and could act as an ancestor of the new activity record through *dis-VisRel*. Thus, in any model of the specification, any ***.Generate*** operation must be preceded an appropriate ***.Create***.

The postcondition that ensures that the new activity records have the right type and are in the right relation with each other is

dis-Po 6: $ActType' \circ (\{a_c,a_p\} \cup A_n \lhd crs\text{-}VisRel' \rhd \{a_c,a_p\} \cup A_n) \circ ActType'^{-1} \subseteq crs\text{-}TypeParent$

where a_c is the new record, a_p is the nearest possible parent activity record, and A_n the set of new activity records (other than a_c). We saw similar postconditions in the operations that create and embed new visit records in the CRS specification (***.EmbInNew*** and ***.EmbInOld***).

An example of an operation that is a refinement of only one from an earlier class is that which is responsible for creating an appointment record:

DIS1Class1.Book$(t_c,pid,c,d,\tau_b \rightarrow a_c,s)$.

This is similar in many ways to the ***.Generate*** operation as it creates a record in *dis-Request* and embeds it appropriate parent records. Because the operation is similar, it is constrained by many of the same preconditions as the operation described above. However, in this case we want to create a booking at the same time. There are two operations that do this in the final class in the OPAS group:

OPASAppt.BookKnownP(pid,sl)

and

OPASAppt.BookUnknownP$(sl \rightarrow pid)$.

The operation ***.Book*** in the DISClass1 class decides which of these two operations to invoke by introducing conditions into the predicates that 'contain' them. ***OPASAppt.BookKnownP*** is only invoked if the patient id that the activity record is associated with is represented in the appointment system's own register: if not, then ***OPASAppt.BookUnknownP*** is invoked. During the operation a slot record is assigned to an appointment record: afterwards, we may need to update the link between the appointment system's register and that of the CRS. Following the operation described here, the pair (*pid,aspid*) will certainly be in *dis-Pidas*, the relation that links the two registers: this does not preclude the possibility that the pair was in the relation before the operation as well. If we have booked an appointment for a patient id in the OPAS' register, then *aspid* is known before the operation is invoked, and there is no change to *dis-Pidas*. If the patient id is not in OPAS' register, then we invoke the second booking operation which returns *aspid*. The way we achieve all this is to define two predicates in the precondition of ***DIS1Class1.Book*** to be:

dis-Pr 22: $pid \in Dom(dis\text{-}Pidas) \Rightarrow aspid = dis\text{-}Pidas(pid) \wedge$ ***OPASAppt.BookKnownP***$(dis\text{-}Pidas(pid),s)$

and

dis-Pr 23: $pid \notin Dom(dis\text{-}Pidas) \Rightarrow$ ***OPASAppt.BookUnknownP***$(s \rightarrow aspid)$

with a postcondition

dis-Po 15: $(pid,aspid) \in dis\text{-}Pidas'$.

An example of an operation that is the composition of several others is the one that is invoked to move the activity record into the set *crs-Proceed* - that is:

DIS1Class1.Start(*a*) where *a* is a known record in *dis-Request.*

We have already seen an operation that puts an activity into *crs-Proceed*: it was introduced in the CRS group of classes and was called ***.CreVis***. The latest refinement of this operation that we are interested in is:

CRSVTClass1.CreVis(*t,pid*→v_n).

In the case of the operation in the composite class, we already know the 'name' of the activity record - it is already a member of *dis-Request*. The purpose of the operation is to take this record and move it to the set *crs-Proceed*. We do not want the 'type' of the record, nor the patient identifier to which it is associated, nor even the identifier of the record itself, to be changed by the operation. For this reason we must be careful to pass it the right arguments. The 'type' of the record before the operation is *dis-ActType*(*a*); that after the operation is the type passed to ***.CreVis*** as the first argument. If the type record that the activity record is associated with is not to change, then we must be sure that this first argument is *dis-ActType*(*a*). A similar presentation could be made for the second argument of ***.CreVis*** which must be *dis-ActSubject*(*a*). Finally, we already know the identifier of the record we are talking about, and we want to make sure that the properties of this activity stay associated with this activity and not given indeterminately to another, and that it is this activity that is moved to *crs-Visit* and not any others. Because of these requirements, the operation inherited from CRSVTClass1 is

dis-Pr 34: ***CRSVTClass1.CreVis***(*dis-ActType*(*a*),*dis-ActSubject*(*a*)→*a*).

By invoking this operation, we have moved a from *dis-Request* to *crs-Proceed*. Any 'parent' activity records that a might have had are also moved to *crs-Proceed*. We know this must be so as a can only have one parent, one 'grand parent', one 'great-grandparent' and so on. ***CRSVTClass1.CreVis***(*t,pid*→v_n) returns $v_{n'}$ with all of its 'ancestor' visit records in *crs-Proceed* - the structure is not changed by invoking the operation, only the status of the activity record a and its ancestors.

The other operation that is inherited here is ***OPASAppt.ApptStart***(*sl*). This puts a time stamp on the slot record, indicating the time of commencement. Not all activity records are associated with slot records and *vice-versa*, The form of the predicate which invokes this operation is thus

dis-Pr 35: $a \in Dom(ActSlot) \Rightarrow$ ***OPASAppt.ApptStart***(*ActSlot*(*a*)).

We might have just as well chosen to represent the pre and postconditions of this operation without referring to these other classes at all. After all, the purpose of a specification is to present to the reader the precise behavioural properties (within certain constraints of scope) of the system being specified *as clearly as possible*. In this case we are specifying the behaviour of a sub-component of the Directorate Information System: there is surely an easier and clearer way to present this behaviour, maybe by defining more 'logical' sub-classes to act as the building blocks of the DIS1Class1 system description? Surely by using 'reverse engineered' descriptions of existing systems as our sub-classes we are forcing ourselves to be unnecessarily constrained by the idiosyncrasies of these earlier systems that were not designed with integration along the lines suggested in mind? This is all certainly true, and the description of behaviour presented here is not the clearest and simplest possible. It does, however, enable us to see how we might set about implementing a system such as that described in DIS1Class1 in terms of the information

systems that we already have. In other words, the reason the description in DIS1Class1 is so complex is that it not only describes (in an abstract way) what the system is to do, but also how (in an abstract way) it might be designed out of existing information systems. It is in this sense that we can say that the specification supports systems integration.

In the next section we will see how the integrated system described here was influenced by the existence of an interaction theory, and consider the way in which we can use such a theory to refine the specification.

12.4 Using An Interaction Theory to Develop an Information System

12.4.1 Introduction

The astute reader should find no great difficulty in getting an idea for the intended 'interpretation' of state components in the specification of the first phase of the DIS. Calling a state component *dis-Activities* immediately tempts us to imagine that this would be designed to represent those quantities in the domain that are called *Activities* in the domain theory: indeed, for the designer of the specification to intend otherwise could be considered mischievous and misleading. The interpretation derived from the names of the state components in this way is informal, probably ambiguous, possibly misleading and certainly not entirely accurate. It is only through the construction of a formal interaction theory that we can gain a proper insight into the role the information system will play in the organisation that has been modelled. Because of this we should endeavour to think of the state components of the information system specification as quantities incorporated within a computer or computers, and it is for this reason that the last section talked of activity records rather than activities.

We have already seen how an interaction theory might be constructed and what it means in Section 11.3. A similar interaction theory has been constructed for the DIS system that was described in the previous section. This is presented in its entirety in Appendix 5. The purpose of this section is not to describe the interaction theory in detail - little would be gained through the completion of this task that has not already been achieved in previous sections (though in a presentation of the specifications to a client, this sort of detailed description might well be required). Rather the author would like to show how the interaction theory was used to guide the development of the specification of the information system - DIS1 - described above.

Before we explore the relation between interaction theory construction and information system specification, a brief discussion of the new techniques that were used in the formal presentation of the interaction theory should be presented.

Firstly, the interaction theory described in the class DIS1Interaction introduces names for state variables of the domain that are supported, or rather represented, by the information system. Thus we have *SupAct* which is the subset of *Activities* that is represented in the DIS1 system. We define this using the assignment

dis-int-I 14': $SupAct = Dom(RepA)$.

We can also give names to subsets of these supported quantities which play a distinctive role in the behaviour of models of the interaction theory. Thus *FullRepTypes* is the subset of *SupTypes* that enumerates a set of types, proceeding activities of which must be supported by the information system.

The assigning of names to these subsets of domain state components makes it easier to describe some of the properties of the interaction theory, and more importantly helps us to understand it.

The interaction theory describes some operations that are not composed of information system operations even though such a composition would be meaningful. This enables us to describe the situation where information system support is available, but is not used for a legitimate reason. For example there are two similar operations in the interaction theory called ***DIS1Interaction.NonRecGenerate*** and ***DIS1Interaction.RecGenerate***. These both invoke the domain operation ***ATClass5.Generate*** to create a new activity and embed it in existing ones - they differ in whether the operation ***DIS1Class1.Generate*** in the information system specification is invoked. The first operation whose name stands for Non-Recorded Generate can be invoked whenever the corresponding domain operation can be invoked and where such an invocation does not effect the state of the information system. The second operation whose name stands for Recorded Generate invokes the pertinent operations from both the domain and from the information system, but only when the type of the new activity is supported (ie it is a member of *SupTypes*). Thus the circumstance where a referral to another health care professional within the DDC is requested but not recorded as such on the information system can be represented.

As mentioned above, we want to be able to represent the legitimate non recording of an operation that changes the state of the domain, but not the illegitimate use. For example, when an activity of a type that is in *FullRepTypes* is started, we intend it to be accompanied by a corresponding operation to 'start' a representation of that activity in the information system. For this reason the precondition

dis-int-Pr 39: $ActType(a) \notin FullRepTypes$

is incorporated into the schema for the operation ***DIS1Interaction.NonRecStart***(*a*). This precondition represents an intended use of the information system - to record the start of every activity of a type in *FullRepTypes*. Of course there is no mechanism available to us that can ensure that the system is used in the manner intended - indeed whenever it is used with 'dummy data' such as during testing it will not be being used in this way - but we are primarily interested in the information system's use rather than its misuse. An example in the DDC is the First Doctor Consultation. It is intended that whenever a patient sees a doctor for an initial consultation, the patient's details should be entered into the computer - the number of such consultation records should thus be equal to the 'number' in the domain. The system is not always used like this however, and some doctors might not bother to enter the patient's details at all, preferring to use paper notes alone. There is nothing that can be done about this, short of encouraging all the users of the system to use it in the manner for which it was designed[xviii].

One, perhaps unrealistic, decision that was taken concerning the interaction theory was to represent the record of a domain state change as happening at the same time as the change being recorded. Information systems are often not used like this. For example many GP systems record details of patient encounters, yet there is no terminal in the consultation room. This is because the doctor writes down the observations made during the consultation in the patient's (paper) notes, and a clerk is employed to enter the new details for all of the day's consultations recorded in the notes onto the computer system at the end of the day.

Such delayed representation of the domain brings problems that an interaction theory would help explore (many of the temporal invariants in the information system would have to be relaxed for example).

[xviii] It should be said here that there is evidence to suggest that users are more willing to enter the data required of them if they know that others do and that they will benefit from it.

However, this thesis has not explored these issues. The formal interaction theory is already a complex entity: the introduction of delayed representation would add a further dimension of difficulty obscuring the lessons learned from the process of constructing and using such a thing. For this reason dissociated domain and information system operations are not described in either of the interaction theories presented in the thesis.

The interaction theory is assembled out of three classes - ClockInteraction, DIS1TypeInteraction and DIS1Interaction. The way these are composed together and with classes form the domain theory and IS theory is given below in figure 3-19.

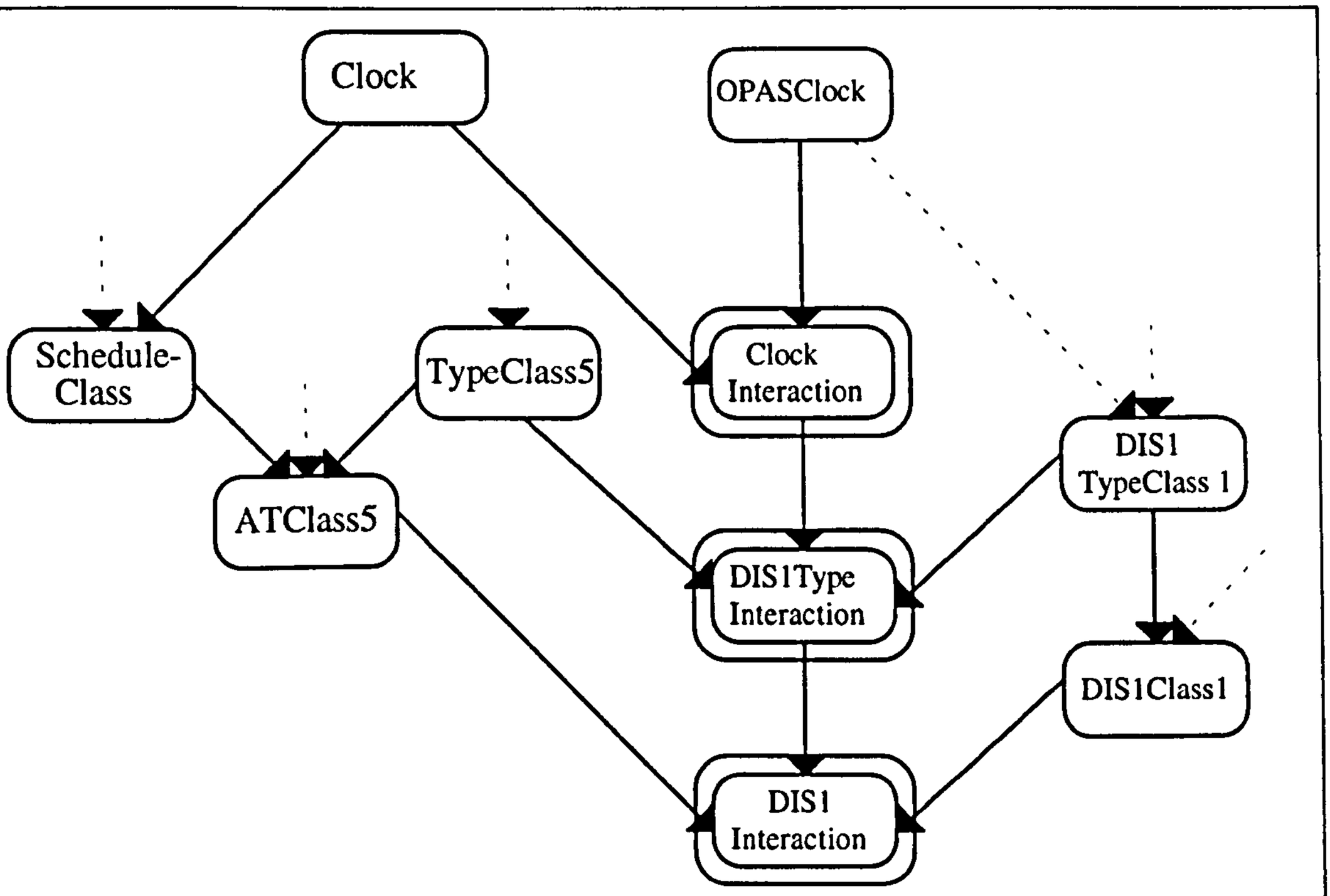


Figure 3-19: Illustration of class structure of interaction theory.

Classes from the domain theory are on the left hand side of the diagram, those from the information system specification are on the right hand side of the diagram, and those that comprise the interaction theory are in the middle of the diagram, distinguished by double borders.

Remember that DIS1Class1 is itself a composition of classes from two earlier theories - the specification of the OPAS and the specification of the CRS. The class which indicates how the proposed DIS is to interact operationally with the domain is the final one: DIS1Interaction.

12.4.2 Four motives to guide development

The ideas and thoughts guiding the development of the information system have been grouped into four sections each of which motivates the decisions taken. Each of these motives is covered in more detail below with the help of examples from the interaction theory we are currently discussing. The four motives are:

- The scope of the information system, as expressed in the interaction theory, should be expanded to cover more of the domain behaviour.

- Each state and behaviour of the information system should have only one interpretation into that part of the domain that is supported by the information system.

- The information system should be specified such that non-sensical operations are prevented as far as possible.

- The information system should not place restrictions on the domain: we should not find that certain operations that were permissible in the domain are prevented through the introduction of the information system.

In all of these motives, we are trying to increase the amount that the information system says about the world. As we saw, we need to make certain assumptions to construct the interaction theory. The interaction theory describes (one possibility of) how the information system will be used, and how it will be interpreted. It must be remembered that it is only within this understanding of how the information system will be used that we can investigate how a better interaction theory might be constructed by changing some of the properties of the information system. If the interaction theory describes a use to which the information system will not be put, then the new information system will not be an improvement at all.

Of course, all of these motives can be entirely fulfilled if the information system is a complete and total representation of the domain. Although the information system could never be a complete representation of the enterprise, we could create one that is a complete representation of the domain theory, which is our best formal description of the 'reality'. In many cases, even this is undesirable. There are many factors that can argue against full and complete representation. The larger the scope of the information system, the more data has to be entered into it to keep the representation accurate and current. This increase in data input presents the user with a greater workload which generally is not wanted. The increase can only be justified if it is necessary to support required system services. What this means is that expansion of the scope of the information system should only occur in areas where support has been requested or is required. In the case of the DIS1 system, we have not represented the name and type of the clinician invoking the operations as this has not yet been requested. Not only might the domain theory have a larger scope than the information system specification, but it might also be more complex. The greater the complexity of an information system, the greater the potential for error. This error might come about in a number of ways: the implementation of the information system is less likely to be a correct model of the specification, errors in the specification are more likely (though these errors would also be present in our domain theory, which we can never be entirely confident about), and greater difficulty in understanding the underlying semantic model might be presented to the user increasing the chance of misuse of the system. Most of these potentials for error can be reduced by the judicious use of formal methods (the power of which has only partly been exploited by the approach recorded in this thesis), but it would be naive to imagine that any of them can be entirely prevented [Goguen90a].

There are a number of other reasons why we might not want close conformity with the domain theory. Firstly, for some of the functions computerised support is not required, and providing it will get in the way. For example, if a patient turns up in the clinic with acute hypoglycaemia, she needs to be stabilised before going into a coma, not recorded on the information system. Secondly there is the need for simplicity to ease the task of system construction and maintenance: the more complex the specification, the more difficult it is to create a computer system that will perform the functions required of it. Simplicity itself is not the only influence on the difficulty of system construction and maintenance: the ease with which the information system can be constructed from existing technical artefacts will have a

direct bearing on the size of the implementation task. These technical artefacts might be the primitive data structures of the implementation vehicle (in this case a relational database and 4GL application description language which readily supports different structures than does, for example, LISP) or legacy computer systems that are to be integrated.

We have seen that there are a number of reasons for the general desire for simplicity in the design of the information system. This has to be balanced by the desire to represent some useful aspects of the domain and to demonstrate conformity with the domain (or at least its descriptive theory) on interpretation. There are a number of ways in which conformity with the domain theory is especially important - these are the four developmental motives which urge us away from simplicity towards 'realism'. The rest of this section will explore the developmental motives listed above in greater detail, with reference to the class DIS1Interaction of the interaction theory.

It should be said before we consider each of these motives in more detail that the business of designing an information system is extremely subtle and complex. That something similar to the interaction theory described can be of help in this process seems clear to the author. Exactly how such a thing should be used is less clear. For this reason the developmental motives represent somewhat artificial categories. We shall see in the last part of the thesis that each of these motives has a theoretical counterpart in computer science where we compare each with an obligation associated with specification reification. In spite of this, the distinction between one motive and another is often not as clear as it would seem from the presentation. Indeed, the same decision might be taken as a result of a number of motives. This is not a problem - we are looking not for categorical neatness in the following discussion, but rather an illumination of how the interaction theory might be beneficially used to tame the enormous difficulty of information system design.

12.4.3 Expansion of scope

This motive encourages us to expand the behaviour space of the specification of the information system to represent a greater area of the behaviour space of the domain theory. This motive is tempered by a desire to have the information system as simple and compact as possible. In general, the greater the degree representation of the domain, the more 'realistic' is the information system, and the greater the degree of support it can provide to the organisation.

The scope of the information system should be expanded in terms of number of state components, number of operations, and the extent of each state component (in terms of the fraction of the domain state component that is represented in the information system). If we look at the interaction theory as it appears in Appendix 5, we can see where this expansion of scope has taken place.

First of all, let us see where it was decided to increase the number of state components. Most of the state components of DIS1Class1 are predetermined by the existing OPAS and CRS systems. Where these state components are to be interpreted as the same domain variable, new functions are defined. Examples of this are: *dis-ActSlot* which relates some *crs-Visit* records with a subset of slots (as well as some *dis-Request* records - the actual type declaration being

dis-17: *dis-ActSlot*: *dis-Activities* ⇸ *as-Slots*);

and *dis-Pidas* which matches the register in the OPAS with that in the CRS. These do not really represent an increase in scope: they exist to mutually constrain the component systems in a realistic manner - something that is discussed further in Section 12.3.5. Two totally new state components have been

introduced that genuinely do increase the system's scope: these are *dis-Request*, which is of the same type as *crs-Visit*, and *dis-Clist*, which is of a totally new type. These expansions are not always as simple as we might think however, as the interaction theory tells us.

Consider first the state component *dis-Clist*. This is defined in the DIS1 information system specification as an aggregation of the set *as-Streams*. This definition was created in such a way that the entity is capable of representing the concept of clinic list that was discovered in the domain. This is the intended interpretation of this state component and the interaction theory says as much. The concept of clinic list has a bearing on the behaviour of slots in the domain. As such it is central to the behaviour of the area of the domain we are endeavouring to support with the proposed DIS1, and represents not only a genuine but also a useful expansion of the scope of the information system.

The state component *dis-Request* also represents a genuine expansion of scope, but is slightly more subtle than in the case of the introduction of *dis-Clist*. We might assume that this straightforwardly represents a state component that has not previously been supported - that is *Request*. The situation is slightly more complex than this as inspection of the interaction theory tells us. Now we know that all complete visit (or activity) records represent complete activities in the domain from the invariant

dis-int-1 22: $SupAct \cap Complete \setminus (im\ IntA)\ Cod(dis\text{-}VisRel) = (im\ IntA)\ crs\text{-}Complete$

from which we can trivially deduce

$(im\ IntA)\ crs\text{-}Complete \subseteq Complete$

(remember that the way in which we constructed the interaction theory was to form the most 'logical' interpretation and then relax it by the minimum amount to fit in with the other invariants and operations - this is how this and the other invariants here were formed). The entity *crs-Proceed* cannot be interpreted so clearly however, and the most constrained invariant we can construct tells us only that

dis-int-1 21: $SupAct \cap Proceed \setminus (im\ IntA)\ Cod(dis\text{-}VisRel) \subseteq (im\ IntA)\ crs\text{-}Proceed$

which is similar to the equivalent invariant in the CRS interaction theory. What this means is that the start of any activity represented by a 'childless' activity record is itself recorded. There might be other activities that have started that are not recorded (even if they are represented activities - a parent activity might well start and not be represented as such), and there might be childless supported activities that are not in *Proceed* that are nevertheless represented as members of the set *crs-Proceed* (these will be activities that have been suspended in the domain). What benefit do we gain by introducing the new element *dis-Request*: that is, how does it expand the interpretational scope of the information system? We will see later on that such an introduction reduces the ambiguity of the information system. But it also genuinely helps to increase the information system's scope. Although inspection of the invariants of the interaction theory does not show how the interpretation of *dis-Request* differs from that of *crs-Proceed*, if we look at the operation schemas of the theory, we see that the only time a record in *crs-Proceed* represents an activity in *Request* is when that activity has been started and then suspended. Thus the set *dis-Request* can represent some part of the domain that was before totally unsupported - namely requests that have never been started.

There are other state components in the domain theory that we have not represented in the DIS1 system. If we look at the final domain theory class - ATClass7 - there are the state components *Followups*, *Records*, *RecSource* and others, none of which is represented in the information system. Earlier in the

theory we were introduced to *In* and *Out* subsets of *Activities*, and to clinicians and other health care professionals represented by the set *HCP*: again, these are not represented in the DIS1 system. These other sets and relations may be useful at some future date, but they do not add greatly to the immediate concern which is to integrate the appointment system with the clinical record system. It is because of this that, by virtue of our desire to avoid any unnecessary complexity, these state components have been excluded. To this end, the interaction theory inherits the earlier class ATClass5 from the domain theory, and even then some of its state components do not have their properties refined in any way as in the case of *HCP*.

We can also see an expansion in scope in terms of the operations supported by the information system DIS1. Examples of explicit operations that are not supported by either the CRS or OPAS systems are ***.Create***, ***.Generate***, ***.SuddenStart***, and ***.Schedule***. We can see by inspecting the interaction theory that each of these operations should be interpreted as its namesake in the domain theory. The operations ***DIS1Class1.Create***, and ***DIS1Class1.Generate*** are not strictly necessary inasmuch as the behaviours of the domain that are of immediate interest to us (those that directly affect the recording of the medical details associated with activities, and those that affect the booking of some of those medical activities) can be understood without recourse to their use. However, the added behavioural richness obtained through the use of the new operations means that we can not only represent the processes we are directly concerned with, but some of the behavioural context within which these are observed. For example the existence of the ***.Generate*** operation enables us to support the creation of requests that are not to be booked for a particular time (at least not yet). Although we do not need to describe this operation to represent booking, its existence lends a greater degree of completeness to the semantics of that part of the information system that covers appointments. The ***.Schedule*** operation is necessary as a result of the introduction of the ***.Generate*** operation - if we did not have the latter, the ***.Schedule*** operation in the interaction theory could always invoke the ***.Book*** operation in the information system.

The third area where we can increase scope is in the coverage of particular state components or operations. For example, we might try to increase the applicability of those operations that create or start activity records in the information system. We could do this by introducing new structures over types and relations over activities so that the limitation on the activity record creation operations is reduced. This would entail effectively recreating *dis-VisRel* as a graph and *crs-TypeParent* as a structure similar in form to *TypeGuide* (we could not recreate what already exists in the CRS, but the addition of extra state components such as those described and their appropriate linking with the CRS components would have a similar effect). This would certainly expand the possible scope of the operations but would vastly increase the complexity of the information system as well as the data input burden of the operator. It is doubtful that the extra value introduced through the creation of these new constructs outweighs the extra complexity of design and use, at least for the time being.

Of course, the decisions involved in choosing whether and where to expand the scope of the information system are entirely subjective, and the criteria used for selecting which additional domain behaviours should be supported verge on the aesthetic. The first motive for development tells us that expansion of semantic scope is generally good in that it increases the amount and richness of information that can be derived from the system. In which cases this motive overrides the counter argument that the information system should be kept as simple as possible in use and structure is open to question, and should be resolved through debate with the users. The interaction theory helps us decide what behaviours there are to be supported, and how that support might be designed.

12.4.4 Functionality of interpretation

This motive ensures that we always understand what any state of the information system seeks to represent in terms of the domain, and similarly what change in the domain any information system operation portrays. The insistence of interpretation of state means that it must be possible to create a total function from any representative state component in the information system to a state component in the domain. We are interested in the representative state components inasmuch as there will be some aspects of the state of the information system that have no bearing on the state of the domain: the set of windows currently open on one workstation, the identity of the button most recently depressed on another workstation, the relation between the identity of users and their passwords and so on. Thus not all information system state components are intended to be interpreted into the domain: still other state components may be only partially so interpretable. Where a state component is partially interpretable, we should be sure that we can determine which members are to be interpreted, and under what circumstances. The same desire for interpretability of individual state components should be extended to the representative state as a whole - thus the entire state of the information system should, as far as possible, be capable of being interpreted as one state of the domain. Although we cannot always ensure this functionality of interpretation and still have the simplicity of design we are simultaneously seeking, it is nevertheless a 'motive' for the development of the system specification.

The desire for functionality pervades the design of the system. Specifically we need to be sure that all the interpretation functions can in fact be functional. This necessitates obvious conformities with the domain theory. For example, the creation of a patient in the information system must be accompanied by the creation of one in the domain, and not several. Suppose that an existing patient id were reused when a patient was registered with the system. If this happened then we could not in general determine which patient a record in the register referred to. Although this example seems so trivial as to be not worth considering, the hospital does re-allocate patient record identifiers in this way if a registered patient does not attend the hospital at any time[xix].

Not only must we be sure that each defined interpretation function is in fact functional, but interpretations of compositions of state components must be as unambiguous as we can reasonably make them. Thus even if all state components can be interpreted functionally, the state of the overall system might be considered to represent any of a number of domain states. Clearly there will always be many possible interpretations of a given information system state not least because many domain state components are not represented, and so can be imagined to take any values. The desire to reduce the un-represented part of the domain was covered by the previous developmental motive. Even when a state component is represented and the state component that is that representation can be interpreted functionally there will still be scope for ambiguity in interpretation of the composite state. Two properties of DIS1 are described below which were introduced to strengthen the system's interpretational functionality, and reduce the scope for interpretational ambiguity of this kind. The first of these properties is that which constrains the relation between the entities *as-Slots* and *dis-Activities*, the second the more specific representation of the domain provided by *dis-Request*.

Inspection of the specification for the integrated DIS1 reveals not only the type declaration

dis-T 7': $dis\text{-}ActSlot: dis\text{-}Activities \rightarrow as\text{-}Slots$

but also the invariant

[xix] This is not as ridiculous as it sounds and is a policy introduced to conserve the pre-printed patient stationery used by the hospital.

dis-I 12: $dis\text{-}ActSlot^{-1} \in Dom(as\text{-}Appointments) \rightarrow dis\text{-}Activities.$

This tells us that every slot record to which a patient id has been allocated is associated with precisely one member of the set *dis-Activities*. Why was it necessary to specify this invariant? Let us consider the situation were we to not have insisted on this constraint. Consider the booking operation. This would allocate a patient id to a slot record, and would create a new activity record. The operation is clear and its interpretation can be functional. As the specification stands, immediately following the operation the slot record must be related to (only) that activity record through *dis-ActSlot*. Without the invariant this relation need not be thus set up: of course we can always insist that such a pair were created but this would specify the same behaviour as might be observed with the invariant, which we have assumed is not necessarily being obeyed.

Immediately after the operation, assuming that the *dis-ActSlot* relation has not been augmented, we lose all knowledge of the relation between the activity and slot records. The user of the system would have no way of linking the newly created activity with the slot. Observing the information system, there would be a number of equally valid interpretations that could be made. For a given slot record we might be able to find a number of possible activities that should be started at that time. Constructing the interaction theory with this reduced version of the information system, we can still make a number of deductions. The activity that the appointment slot record refers to must have the interpreted patient id as its subject, it must be in *Request*, and it must be of a bookable type. Other than that we cannot assume anything about the activity. There might be a number of activity records in the information system that fulfil these criteria, and any one of them might be the one that the appointment record concerns. In short, there are a number of interpretations that could possibly be made of the state of the information system, one for each pairing of the interpreted slot record with a valid (represented) activity.

The only way of recording which slot relates to which activity would be to write it on a piece of paper: not a useful suggestion when we are talking about the design of a future automated information system. Even if such redress were taken, the same problem would be encountered when the activity started. As the information system doesn't record which is the associated slot record, and it is to this latter entity that the time (that the *.Start* operation is invoked) is attached, the user could, after the operation, only derive a similarly ambiguous understanding of the start time of the activity. In short there is no way of determining (from the information system) with certainty the start time of any activities.

There are thus several problems relating to this ambiguity of interpretation. We don't know when a booked appointment is to start: we can send a letter to a patient requesting them to attend the clinic on a specific day, but could not then (in general) decide which clinical encounter they had arrived for. Similarly in future we could not determine when the activity had started, and when it had finished. All these problems are solved by insisting on an invariant over the integrated system of the form

dis-I 12. $dis\text{-}ActSlot^{-1} \in Dom(as\text{-}Appointments) \rightarrow dis\text{-}Activities.$

Now, when we book a patient in for a particular clinical activity, the identity of that activity is recorded in the information system. When we start the activity, we can ensure in the information system that the slot starts at the same time. The start and finish times of a given (booked) activity can thus be recorded. These latter observations are recorded in the two interaction invariants

dis-int-I 29: $as\text{-}StartTime \subseteq RepTime \circ (SupSlot \lhd (ActStart \circ ActSlot^{-1})) \circ IntSlot$

and

dis-int-I 30: $as\text{-}EndTime \subseteq RepTime \circ (SupSlot \lhd (ActEnd \circ ActSlot^{-1})) \circ IntSlot$

which say that the start and end times associated with slot records can be functionally interpreted as start and end times of activities.

A similar (though less interesting) argument to the above can be presented justifying the linkage of the two patient registers *crs-Pid* and *as-Pid* through the function *dis-Pidas*.

We should note here that the important feature of $dis\text{-}ActSlot^{-1}$ was the totality of the relation, not its functionality. In fact, the functionality places a constraint on the information system limiting the support it can give to the organisation. Recall (Section 10.2.4) that the assumption that one activity could only be associated with one patient was challenged by the example of the patient education session. The same example presents problems for the functional relation between the slot record and the activity record. If the patient education session is to be represented as many parallel activities, then the slot that is appropriate for the patient education session must similarly be related to multiple activities. As this is prevented by the integration invariant we have seen, patient education sessions can not be supported by the DIS1 (at least not straightforwardly - there are ways around this problem such as the creation of many sequential slot records of very short duration, but this leads to other problems). The integration invariant does not constrain the system further than it would do if the inverse of *dis-ActSlot* had been relational as the following type declaration in the OPAS system

as-T 24: $as\text{-}Appointment: as\text{-}Slots \twoheadrightarrow as\text{-}Pid$

makes clear.

One place where existing invariants meant that the functionality of interpretation was improved is in the introduction of *crs-Requests*. We know that the state component *dis-Activities* can be interpreted as a set of *Activities* in the domain as illustrated by the following type declaration from the class DIS1Interaction:

dis-int-T 5: $IntA: dis\text{-}Activities \rightarrow Activities.$

Thus for each member *dis-a* of *dis-Activities* there is a corresponding member a of Activities that is its interpretation in the domain. The same cannot be said for the subset of *dis-Activities* called *dis-Request*. There is no function in the interaction theory that interprets *dis-Request* as we do not know, without inspecting the state components of the domain explicitly, whether the members of *dis-Request* should be interpreted as members of *Request* or *Proceed*. This is because activities in the domain can start without that fact being recorded on the information system: there is no way of interrogating the information system, even if it has been used correctly, to determine whether a member of *dis-Request* is supposed to represent a requested or proceeding activity. The same is true of the information system state component *crs-Proceed*. Although we know that there is an interpretation of this state component as a subset of *Activities*, ie

(*im IntA*) $crs\text{-}Proceed \subseteq Activities$

(which we can deduce from *crs*-I2, dis-I5, and dis-int-T5)

we do not know what that subset is (although from dis-int-I21 we can see that there is an identifiable subset of (*im IntA*) *crs-Proceed*). An activity in the domain might be started, and have that start recorded in the information system, but might then be suspended and become a member of *Request* again: there is no information system operation that represents the ***.Suspend*** operation in the domain. Similarly only members of the *crs-Proceed* set that have no 'children' though *dis-VisRel* can be completed in DIS1 so we cannot say that a member of *crs-Proceed* does not represent a completed activity. However the situation is not as vague as it might be - we can say that any member of *crs-Complete* represents a completed activity, and that no member of *dis-Request* represents a complete activity.

We know that the first of these statements must be true by investigating the operations that have been defined in the interaction theory. As described, the intended usage of the system is such that the only time an activity record in DIS1 is completed, the corresponding activity in the domain is also completed. This is represented by the following schema

DIS1Interaction.End(a)

dis-int-Pr 50: $a \in SupAct \Rightarrow \boldsymbol{DIS1Class1.End}(RepA(a))$

dis-int-Pr 51: $\boldsymbol{ATClass5.End}(a))$

which says that the information system operation is invoked whenever there is a representative of the activity being finished in the domain, and the preconditions for the invocation of the information system operation are satisfied. Note that one of the (inherited) preconditions of ***DIS1Class1.End*** is that *RepA*(a) is a childless activity record. The absence of a ***.NonRecEnd*** operation reflects one of the intended usages of the information system: namely that the completion of all activities in the domain that are represented in the information system by a 'childless' visit record must be recorded in the database. This intended usage is described in the interaction theory by the invariant

dis-int-I 22. $(SupAct \cap Complete) \setminus (im\ IntA)\ Cod(dis\text{-}VisRel) = (im\ IntA)\ crs\text{-}Complete.$

Whatever the circumstances for invocation of the information system operation, we can see from the interaction theory operation defined here that the ***.End*** operation is always invoked in the domain: whenever an activity record in completed in the information system, it must be completed in the domain also. This simple observation enables us to write

$(im\ IntA)\ crs\text{-}Complete \subseteq Complete$:

that is, that all members of *crs-Complete* should be interpreted as completed activities is a straightforward implication of the intended use of the system.

The situation with *dis-Requests* is a little more complex, but from the existing interpretations and invariants in the information system specification we can see that such a record, although possibly representing a proceeding activity, should never be interpreted as a member of *Complete*. We can show this is so from the following argument.

Recall that in the specification of DIS1 we not only introduced the set *dis-Request*, but also expanded the tree over the (similarly expanded) set *dis-Activity* to form the new relation

dis-T 4: $dis\text{-}VisRel: dis\text{-}Activity \nrightarrow dis\text{-}Activity$

where

dis-I 8': $crs\text{-}VisRel \subseteq dis\text{-}VisRel$.

The invariant

dis-I 11: $Dom(dis\text{-}VisRel \setminus crs\text{-}VisRel) \subseteq dis\text{-}Request \wedge Cod(dis\text{-}VisRel \setminus crs\text{-}VisRel) \cap crs\text{-}Proceed \subseteq Cod(crs\text{-}VisRel)$

describes the difference between *dis-VisRel* and *crs-VisRel*. All activity records that are the first half of a pair which is in *dis-VisRel* but not in *crs-VisRel* must be members of the *dis-Request* set. This is because as soon as an activity record starts (ie - moves into *crs-Proceed*), then its 'parent' must also be in *crs-Proceed*. *dis-VisRel* is intended to be the smallest extension of *crs-VisRel* needed to represent inclusion in the new activity records. The difference between it and *crs-VisRel* thus contains pairs of the type (request, request), (request, visit), and (visit, request) of which the last is not possible as we have seen. The second part of the invariant says that if any second part of a pair in this set difference is a visit record, then that visit record must also have a 'child' as recorded in *crs-VisRel*. This is because only activity records that are of types that are forbidden 'children' via *crs-TypeParent* can start explicitly. In other words any activity in the codomain of *dis-VisRel* that has started must have started because one of its 'child' activities started: a pair consisting of the started parent visit record and the necessary started child visit record will be in *crs-VisRel*.

dis-int-I 16: $IntA \circ dis\text{-}VisRel \circ RepA \subseteq During$

tells us that the *dis-VisRel* tree represents part of the *During* graph. Finally, we know that

19: $(im\ Includes)\ Complete \subseteq Complete$

from the domain theory.

These invariants together with the one defining intended use that we saw earlier, namely

dis-int-I 22: $SupAct \cap Complete \setminus (im\ IntA)\ Cod(dis\text{-}VisRel) \subseteq (im\ IntA)\ crs\text{-}Complete$,

show that a member of *dis-Request* can never be interpreted as a completed activity. We demonstrate this through the use of the principle of *reductio ad absurdum* where we assume that the property to be proven is false and show that this leads to an inconsistency.

Suppose that a member of *dis-Request*, *dis-r*, were interpreted as a member of *Complete*, c, then we would have

$dis\text{-}r \in dis\text{-}Request$, $c \in Complete$, *and* $(dis\text{-}r, c) \in IntA$.

dis-r is either in the codomain of *dis-VisRel* or it is not. We will take the two cases separately. Suppose

$dis\text{-}r \notin Cod(dis\text{-}VisRel)$

then, as *IntA* is a function we can say that

$c \notin (im\ IntA)\ Cod(dis\text{-}VisRel)$.

But

$c \in Complete$, *and* $c \in SupAct$

because

$SupAct = Cod(IntA)$.

We can thus say that

$c \in (SupAct \cap Complete) \setminus (im\ IntA)\ Cod(dis\text{-}VisRel)$.

From Invariant dis-int-I22 we can hence infer that

$c \in (im\ IntA)$ *crs-Complete*:

that is, the representat'on of c must be in *crs-Complete*. In this case the interpretation of the request record as a completed activity is inconsistent. Suppose we take the other case where

$dis\text{-}r \in Cod(dis\text{-}VisRel)$.

As the set of *dis-Activities* is finite in extent, and as *dis-VisRel* is directed, there must be a descendant of dis-r that is not in *Cod*(*dis-VisRel*): ie

$\exists desc_{dis\text{-}r}$: $(im\ dis\text{-}VisRel^{-1+})\ \{dis\text{-}r\} \bullet desc_{dis\text{-}r} \notin Cod(dis\text{-}VisRel)$.

Because of the invariant showing that we can interpret *dis-VisRel* as part of *During*, we know that there is an interpretation of $desc_{dis\text{-}r}$, call it $desc_a$, where

$desc_a \in (im\ Includes^{+})\ \{c\}$.

Invariant I9 tells us that the child of a complete activity is complete: so its grandchild must be too, and hence its great-grandchild, and so on. Thus from this invariant and the previous result we can deduce that

$desc_a \in Complete$.

The first part of the argument showed us that if an activity is completed, and it is represented by an activity that is not in the codomain of *dis-VisRel*, then that representation cannot be in *dis-Request*: it must thus be in *crs-Visit*. The record $desc_{dis\text{-}r}$ is not in *dis-Request*, but one of its ancestors, *dis-r*, is. The information system invariant dis-I11 showed that

dis-I 11: $Dom(dis\text{-}VisRel \setminus crs\text{-}VisRel) \subseteq dis\text{-}Request \wedge Cod(dis\text{-}VisRel \setminus crs\text{-}VisRel) \cap crs\text{-}Proceed \subseteq Cod(crs\text{-}VisRel)$

from which we can infer (for a directed acyclic tree which is what $dis\text{-}VisRel \setminus crs\text{-}VisRel$ is) that

$Dom((dis\text{-}VisRel \setminus crs\text{-}VisRel)^{+}) \subseteq dis\text{-}Request$.

The pair ($desc_{dis\text{-}r}$, *dis-r*) must be a member of $(dis\text{-}VisRel \setminus crs\text{-}VisRel)^{+}$ as *dis-r* does not feature in *crs-VisRel*. This again is inconsistent - we demand at once that $desc_{dis\text{-}r}$ is a member of *dis-Request* and *crs-Visit* where

$dis\text{-}Request \cap crs\text{-}Visit = \emptyset$.

We have thus shown that no record in *dis-Request* can be interpreted (according to this interaction theory) as an activity in *Complete*.

In this section we have examined a couple of areas where the second developmental motive helped guide design decisions. The motive encouraged us to ensure that the information system could be interpreted into the domain functionally. This applies not just to individual state components but to states represented by compositions of several state components. The examples discussed above demonstrated how judicious use of state components and invariants in the information system could reduce possible ambiguities of interpretation (assuming that the system was interpreted and used in the way specified by the interaction theory). The first example concerned the specification of an invariant linking the domain of *as-Appointments* with records in *dis-Activities*. Through the introduction of this invariant property in the information system design, by inspecting the information system we can determine which slot relates to which activity: without it we would not know (through the information system) precisely which activity should be started when a particular slot was due to commence. The second invariant introduced concerned the nature of the extension to the relation *crs-VisRel* in DIS1 (that is *dis-VisRel*). The existence of this invariant (and the operations that support it) means that although we cannot be sure that a member of *dis-Request* does not represent a proceeding activity, we do know that it can never represent one that is in *Complete*. By reducing the ambiguity (and thus increasing the functionality) of possible interpretation, we have effectively increased the quantity or quality of information deducible from the system.

12.4.5 Restriction of non-sensical IS operations

This is one of the most important of the four motives as it helps us to make decisions concerning the extent of the information system, especially when it comes to system integration. This motive helps us introduce three sorts of constraint. The first is a constraint on an existing information system which previously supported non-sensical operations. For example there are a number of operations supported by the Clinical Record System that are not observable in the domain. An example of one of these spurious operations is the one in the CRS which allows many activity records of the same type, with the same parent activity record, to be in *Proceed* at the same time - when this is non-sensical and how we refine the system to deal with the problem is described below. The second sort of constraint restricts the values of a state component that has been introduced in this system (this is the only sort that applies if the system is totally new and not an integration of any others). An example of such a new state component is the introduction of a representation of the clinic list: various invariants are needed to prevent non-sensical states such as the existence of two overlapping slots in the same clinic list. This example is discussed below. The third sort of constraint is most interesting: these constraints ensure that the component information systems of an integrated system interfere with one another and mutually restrict each other's possible states. Before we consider examples of these sorts of constraints and the non-sensical behaviours and states they prevent, let us consider this last case and see how the introduction of such mutual constraints is the essence of system integration.

Consider a system S1 which has a number of possible behaviours. These behaviours correspond to trajectories in the system's 'state space'. The number of possible trajectories will be huge, but assuming that our carrier sets are finite, the state space will be finite, and so will be the number of distinct trajectories. Let us call this number of possible behavioural trajectories of S1 N_{S1}. In the same way, we can call the number of possible behavioural trajectories of some other system, S2, N_{S2}. Suppose we somehow combine the two systems S1 and S2 to create a third composite system S3. If S1 and S2 are

totally un-integrated, the number of possible behavioural trajectories is $N_{S1} * N_{S2}$. This is the case when the state of one information system is totally independent of the state of the other.

Suppose we now take two aspects, or sub-domains, of some domain of interest: if these two sub-domains interact with each other at all, then the behaviour space of the domain is smaller than the product of the behaviour spaces of its sub-domains. For example, in the case of the domain of the DEDC, we can look at the sub-domains of appointment booking and activity delivery. Here, the two sub-domains interact, and in our theory, we cannot book an appointment after we have completed the activity to which that appointment refers: this is an example of a class of behaviours that is forbidden to the domain by virtue of their interaction.

Suppose our two information systems, S1 and S2, were representations of two interacting aspects of the same domain. If S1 and S2 were reasonable representations, then it would be likely that an un-integrated S3 would support behaviours that were not seen in the composite domain. An integrated information system is one that consists of component systems that represent different, but interacting, sub-domains of a single domain in such a way that those behaviours that are prevented in the domain by the interaction of the sub-domains, are also prevented in the information system by the constraints placed on the composition of the component systems. What this means is that if S1 were an appointment booking system (for example the OPAS described above), and S2 were an activity delivery system (for example the CRS), then an integrated information system that supported both activity delivery and appointment booking would, unless S1 and / or S2 were very deficient in their own support for their respective sub-domains, have a much smaller number of possible behavioural trajectories than $N_{S1} * N_{S2}$. This is the essence of systems integration - preventing those behaviours of a composite information system that are not observed in the domain by virtue of the interaction of the sub-domains that each system component represents.

The above discussion begs the question 'why bother?'. Why should we be interested in preventing the user from accessing operations that correspond to behaviours that are never seen in the domain. Surely the user will never have need to invoke such operations, so their existence or lack thereof is irrelevant? In the case of the integration of two systems the reason for the explicit representation of the mutual constraints that the two sub-domains place on each other is clear: the un-integrated approach will result in the need for more work from the user in order to represent a given change in domain state. If a single event in the domain is represented in both S1 and S2, then a user will have to invoke two operations - one for S1 and one for S2 - if the composite information system is to represent the domain. By representing this operation in the domain as one (integrated) operation in the information system, we cut down on user work, but prevent the independent alteration of the state of S1 and S2.

It is in fact the reduction of work from the user that is the background to this motive. For any state of the information system, the user is presented with a set of potential behaviours from which one must be chosen. The smaller this set the easier the choice. If the magnitude of the set is increased through the presence of behaviours that should never be selected, the task of the user is rendered more onerous needlessly. We can see this more easily if we consider an example from a field other than computing.

Consider the task of the driver of a car when steering the vehicle: by the introduction of a constraint in the form of a steering rod the number of possible behaviours that a car can exhibit is reduced, but the driver's job is greatly eased. The front two wheels of a back wheel drive car are in many senses independent of each other. Each wheel revolves independently at whatever speed is appropriate for the local ground speed: each moves vertically in the shock absorbers independently, responding to the bumps

in the road. In at least these respects, the two wheel assemblies are un-integrated (at least explicitly by the car designer - in use each wheel revolves at approximately the same speed, the integration being provided by the single surface they are running on). However, in terms of lateral rotation - steering - the wheels are integrated by the steering rod that connects them. In the absence of a steering rod, we could still have reliable control of the position of the car if we were provided with two steering wheels. However, driving would be much more difficult as we would have to be careful that we turned the steering wheels in concert otherwise the car would skid and possibly crash. The introduction of two steering wheels would not prevent us from driving properly, but it would make it much more difficult. The reason for this is that we would be presented with many more possible 'behaviours' to choose from. Most of these behaviours result in 'non-sensical' behaviour (in this case an angle between the two front wheels leading to a skid). Through 'integrating' the steering of the two front wheels, we drastically reduce the number of possible behaviours that the driver can choose from. This is not an arbitrary reduction - there are many integrations that would be disastrous. For example, we could say that when the left wheel turned to the right, the right one turned to the left. This would present the driver with a behaviour space the same size as that of the system when it was constrained by the steering rod, only this time almost all of them would be 'non-sensical'.

Similar arguments to those presented above hold in the case of information systems development. We must present the user with the smallest number of choices necessary to record the change in state of the domain, or at least that part that is supported by the information system. What we are doing in effect is to reduce the entropy of the system, without reducing its informational content. This motive does not just apply to systems integration: it is vital to all types of system development, for as we extend an information system, we are in effect integrating sub-components into a whole, although we might not explicitly recognise that this is the case. The desire to restrict non-sensical operations is essential to systems design, and is the essence of systems integration. Remember again, however, that this motive should be tempered by the need to keep the semantic model embodied in the information system simple, concise and robust, meaning that we should think very carefully if a highly complex representational structure were introduced which only prevented a very small number of behaviours. Indeed, on the whole we will be powerless to prevent the vast majority of non-sensical behaviours: when we are presented with an opportunity to decrease the entropy 'painlessly' we should seize it.

How do we recognise a non-sensical operation? How can we tell if the information system allows an operation that is not valid in the domain? This is not difficult, as long as we construct the interaction theory carefully. A non-sensical behaviour is one that the information system can exhibit but which does not represent one in the domain. If we consider a state of the interaction theory, this will have corresponding information system and domain states (along with the interpretation functions relating those two states to each other). If a particular change to the state of the information system is allowed by the information system specification but prevented by the interaction theory then this is an example of an illegal behaviour. Of course we are less concerned with individual examples of such behaviours than with classes of them. The reason we must construct the interaction system carefully is because we must ensure that each operation corresponds to one in the domain theory, and there are no cases where the invocation of an interaction theory operation invokes an information system operation but not a domain theory operation. Inspection of the interaction theory for DIS1 reveals that this is the case here. A further sophistication, discussed in section 14.7, would be to allow the state of the information system to 'lag behind' that of the domain. This would represent the way that many information systems actually work to represent an organisation. Although the interaction theory in this case would be significantly more complex than that presented here we would still have to be sure that what was being reported (albeit in a post facto manner) was a valid state of the domain.

The vast majority of design decisions taken in the construction of the DIS1 specification were of this entropy reducing nature. This is not surprising as DIS1 is an integration of two subsidiary systems - OPAS and CRS - each supporting and thus representing separate sub-domains: as the sub-domains interact and thus constrain each other, we would expect there to be many invariants in the composite system acting to mirror those constraints. However, not all the constraints specified in the composite system act to integrate the sub-systems. Indeed, there are a number of ways in which either of the components are representationally inadequate prior to integration, and where each supports non-sensical operations even in its own sub-domain. The introduction of invariants in the integrated system might improve one of the individual component systems independently of any other parts (if there are any) of the complete system.

We will discuss four cases where the introduction of invariants or state components acted to reduce the entropy of the system. Firstly we shall consider the introduction of an invariant which constrained the CRS independently of the OPAS. Secondly we shall consider the way in which constraints needed to be created following the introduction of a new state component. Thirdly we shall see how the fact that DIS1 is an integration of two component system led to the introduction of a mutual constraint. Fourthly we shall see a more complex argument that justifies the increasing of the scope of a particular state component in terms of the reduction of entropy.

The first example concerns the introduction of an invariant that constrains state components from the CRS system alone. As such it is not an aspect of the systems integration that is essential feature of the DIS1: it is an 'improvement' to the CRS. An invariant noticeably lacking from the CRS specification is one that prevents two visit records of the same type with the same parent record from being in the set *crs-Proceed*. An invariant over the state components of the domain theory that are the interpretations of these sets does exist that mirrors this property, namely:

I 26: $\#ActType \circ (Includes \rhd Proceed) = \#(Includes \rhd Proceed)$.

The absence of a corresponding invariant in the CRS leads to the possibility of non-sensical behaviours. Two visit records of the same type with the same parent cannot be in *crs-Proceed* simultaneously unless they are of a type that can have no children. This is because such visit records cannot be created explicitly at all, but are rather recorded as a result of the explicit 'starting' of records of a type that has no children, as stated in the precondition to ***CRSVTClass1.CreVis***:

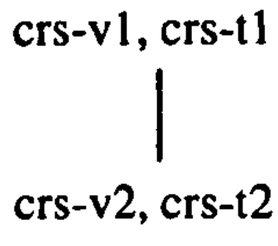
Pr 22: $t \notin Cod(crs\text{-}TypeParent)$.

If a visit record in *crs-Proceed* exists that could possibly be a parent of the newly created record, then after the completion of the operation it will be and a new parent visit record will not be created. However, the same is not true of those visit records of types that do not have any children through *crs-TypeParent*. There is nothing to prevent the creation of any number of such visit records of the same type with the same parent, all in the set *crs-Proceed*. This clearly can give rise to non-sensical states of the information system. We know from the interaction theory invariant that

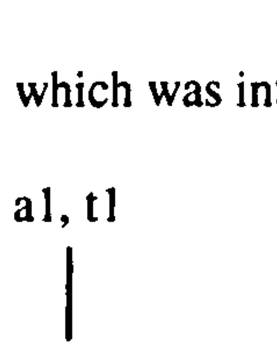
crs-int-I 7: $((im\ IntV)\ crs\text{-}Proceed) \setminus Cod(crs\text{-}VisRel) \subseteq Proceed \cup Request$.

In other words, a visit record such as that which we are considering (that has no children) is to be interpreted as either being a request (as long as that request is a suspended proceeding activity) or a proceeding activity. Now we know from the domain theory that there can only be one activity of a given type with the same parent activity that is in *Proceed* or was in *Proceed* but has since been cancelled.

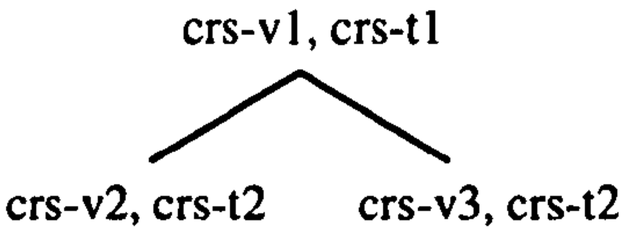
From this we can see why the operation to create two members of the *crs-Proceed* set represents an illegal behaviour in the domain. Suppose we had a state of the information system represented by the following graph:



which was interpreted as the following structure in the domain:



We know that if *crs-v2* is in *crs-Proceed*, then *a2* must either be in *Proceed*, or have been in *Proceed*, but since been suspended into *Request*. If we take the information system specification on its own, then an operation which creates a new visit record of type *crs-t2* is not prohibited. The resulting state of the information system would be represented as



again where both *crs-v2* and *crs-v3* are in *crs-Proceed*. This new state in the information system would be prohibited in the interaction theory however for the following reason. If *a2*, the interpretation of *crs-v2*, were in *Proceed*, then we would have two activities of the same type with the same parent in *Proceed* which is prohibited by invariant I26. If it were in *Request*, then another proceeding activity of that type could not be started by virtue of precondition Pr67 (to the *.SuddenStart* operation) and Pr71 (to the *.Start* operation). We have seen a class of cases where, starting with a valid model of the interaction theory, the model of the information system in isolation allows behaviours that are forbidden in the model of the interaction theory. These are the non-sensical behaviours described in this section, and if they can be prevented then they should be.

In the event this class of non-sensical behaviours is prohibited by the introduction of the invariant in DISClass1:

dis-I 21: $\#(crs\text{-}Proceed \lhd crs\text{-}VisPid) = \#(crs\text{-}Proceed \lhd crs\text{-}VisPid) \circ crs\text{-}VisitType^{-1}$.

We are able to make the invariant of this form (no two visit records with the same patient id, rather than with the same parent) because if two visit records have the same patient id associated with them, then they will have the same parent visit type. This is true because *crs-TypeParent* is a function: two visit records of the same type must have the same type of parent, and by virtue of the above invariant, as those parents must be in the same subset of *crs-Visit* (ie *crs-Proceed* or *crs-Complete*) they must be the same record.

The second example of the introduction of invariants to reduce entropy relates to the introduction of a new state component in the information system. The state component *dis-Clist* was introduced to represent clinic lists in the domain. this is recorded in the interaction theory through the interpretation function:

dis-int-T 8': $IntClist$: $dis\text{-}Clist \rightarrow C$

(remember that we did not trouble to consider the creation of clinic lists in the domain which is why it was left as a carrier set). The introduction of this state component, and its intended interpretation, gives rise to the need for a number of invariants to prevent potentially non-sensical operations. The problem comes from the relation between slots and clinic lists. In the information system, a member of *as-Slots* is functionally linked to an entity called *as-Stream*. *as-Slots* is interpreted onto the state component *Slots* through the interpretation function **IntSlot**. There is no direct interpretation of *as-Streams* however, rather members of this set are functionally linked to the state component *dis-Clist*. If this were as far as the DIS1 specification went then the possibility of non-sensical behaviours arises. *as-Slots* records that are associated with the same *as-Stream* must not 'overlap' (by virtue of their representation of their 'start' and 'stop' times). As several members of *as-Stream* can be associated with one *as-Clist* record however, two records in *as-Slots* that are linked to the same *as-Clist* record might well be overlapping. The domain state that such an information system state attempts to represent is illegal. We exclude the representation of these impossible domain states through the introduction of the invariant

dis-I 14: $(dis\text{-}StreamClist \lozenge as\text{-}StreamDay)^{-1} \in (dis\text{-}Clist \times as\text{-}Day \nrightarrow as\text{-}Stream)$

which insists that only one *as-Stream* record per *as-Day* can be associated with a *dis-Clist* record. As as-Day is the representation of a 'day' (defined in the class ClockInteraction - there is no 'day' in the domain theory), this effectively prevents overlapping *as-Slots* records.

Note that we could have chosen not to introduce a representation of the clinic list at all, and said that *as-Slots* should be interpreted as clinic lists instead (although the inverse of such an interpretation would not have been functional): that is

dis-int-I 7': $IntSlot$: $as\text{-}Slots \rightarrow C$.

If we accompanied this with a statement of intended use (ie an invariant in the interaction theory) which said that any two slots interpreted as the same element of C must be associated with separate days. This alternative course of action was not taken as it was felt that clinic lists were entities with a greater degree of 'reality' in that clinicians spoke about these in interviews, and did not explicitly mention the concept of streams.

The third example of a change introduced by virtue of this motive concerns the integration of the two subsidiary systems CRS and OPAS to form DIS1. The introduction of an invariant constraining the OPAS and the CRS was discussed in the section on the functionality of interpretation. This decision also has implications for the prevention of non-sensical operations. Here we will consider the integration of two information system operations however.

If we look at the .Start operation in the class DIS1Class1, we notice two things of interest. The first point to note is that one operation in the integrated information system invokes two others - one from each of the composite systems. The second point to note is that whereas the two component operations take three arguments between them, the composite one takes only one. The latter point is a consequence of the introduction of the set *dis-Request* and is of the same type of 'entropy decreasing' measures as were discussed earlier in this section. It is the first point that we will discuss now.

As with the first part of this section, we will first consider what behaviours would be available for a user to invoke if the system were not 'integrated' in the way that we have specified it to be. Suppose that the

single operation ***.Start*** were replaced with two others that were capable of being invoked independently. These would be ***.CreVis*** which invokes the eponymous operation from the class CRSClass4, and ***.ApptStart*** which runs its namesake from the class OPASAppt. While we are tinkering with the interaction theory, we had better be sure we know how the different concepts are to be interpreted. In the interaction theory of the integrated system as represented by DIS1Class1, members of Dom(*as-Appointments*) can be interpreted as members of the set *dis-Activities*. Although this interpretation is not represented explicitly, it can be derived through the function *dis-ActSlot* and its inverse. The function *dis-ActSlot* is one of the ingredients of the system integration that we are now assuming is missing - for this reason we cannot presume its existence, and cannot thereby deduce the interpretation of *Dom*(*as-Appointments*) as a subset of *Activities*. This does not alter the meaning of the set - we still want *Dom*(*as-Appointments*) to be interpreted as a set of *Activities*, but must now state this explicitly in the interaction theory. We would thus expect to see that *Dom*(*as-Appointments*) could be so interpreted, and must thus state that this is its intended interpretation in the interaction theory.

As we saw above, a good way of seeing whether non-sensical operations are allowed in the information system is if the interaction theory has to impose constraints on the information system. This would be the case here. The interpretation of ***.CreVis*** is the domain operation ***.Start***: in other words, the ***.CreVis*** operation would be invoked by an interaction theory operation that invoked the domain operation ***.Start***. The problem is that the same would be true of the operation ***.ApptStart***. The operation in the interaction theory would look something like this:

DIS1Interaction.RecStart

[Types]
[Preconditions]
[Predicate 1 ⇒] ***DIS1Class1.CreVis***
[Predicate 2 ⇒] ***DIS1Class1.ApptStart***
ATClass5.Start
[Postconditions]

Predicate 1 is some predicate that must be satisfied if ***DIS1Class1.CreVis*** is to be invoked, and Predicate 2 if ***.ApptStart*** is to be invoked. The important thing to note is that there will be times when both predicates are satisfied and on invocation of the interaction theory operation, the two information system operations are invoked together: and it is at these times that the system will present the user with non-sensical operations. For example, the user might choose the ***.CreVis*** operation without picking ***.ApptStart*** to record the start of a booked activity. It would take two separate invocations of information system operations to represent one domain operation. The state of the information system between these operations would thus not represent the domain.

The solution to this problem as expressed in the version of the DIS1 specification found in Appendix 5, is the explicit representation of the link between some members of *as-Slots* and some members of *dis-Activities*, and the combining of the two ***.Start*** operations into one. Now, not all activities that start have been booked, and we would want to reflect this in the integrated system. We thus embed the invocation of the ***OPASAppt.ApptStart*** operation in an implicative predicate. Thus the ***DIS1Class1.Start*** operation

always invokes the ***CRSClass4.Start*** operation, but invokes ***OPASAppt.ApptStart*** only when the slot in question is a booking, or a member of *Dom*(*as-Appointment*). The operation thus looks like

DIS1Class1.Start(*a*)

dis-Pr 33: a: *dis-Request*

dis-Pr 34: ***CRSClass4.CreVis***(*dis-ActType*(*a*),*dis-ActSubject*(*a*)→*a*)

dis-Pr 35: $a \in$ *Dom*(*dis-ActSlot*) ⇒ ***OPASAppt.ApptStart***(*dis-ActSlot*(*a*))

Now that we have linked the two operations together in this way, the non-sensical behaviours that were described earlier have been excluded. We have thus reduced the system entropy in a desirable manner.

The last example of the influencing of a design decision as a result of this motive is a subtle one. We could have chosen to introduce the new entities d*is-Request* and *dis-Activity* without expanding the scope of the various functions of *crs-Visit*. This might well have been simpler in certain implementations of the system. However, this motive encouraged the decision to introduce new state components that mirrored the attributes of *crs-Visit* in the CRS. Let us consider one of these, *crs-Visit*Type and its expansion to *crs-ActType*.

If we are to understand why *dis-ActType* was necessary, let us imagine how the DIS1 system and the interaction theory would have looked were we to have excluded it, and relied on *crs-Visit*Type to tell us what type a given activity was. The function *crs-Visit*Type returns the type record associated with any activity record, stored in the CRS, that is a member of the sets *crs-Proceed* and *crs-Complete*. We know this from the type declaration

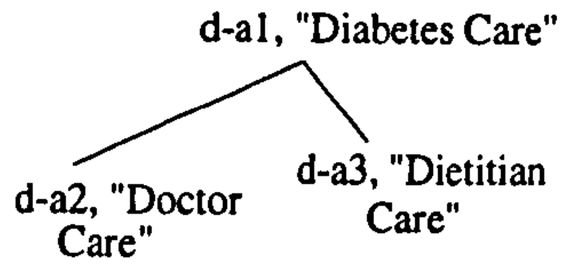
crs-T 7: *crs-Visit*Type: *crs-Visit* → *crs-Types*

and the invariant

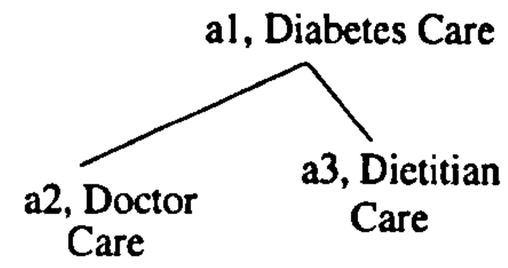
crs-I 2: *crs-Proceed* ∪ *crs-Complete* = *crs-Visit*.

In other words, records in the set c*rs-Request* would not be associated with type records. This will have one immediate effect on the operations that we can specify on the information system. When we invoke the ***.Generate*** operation, because we cannot allocate a type to a request record, we cannot determine which is the appropriate 'parent' activity record. When invoking the ***.Generate*** operation we must specify the correct parent activity record - this in turn means that an immediate parent for the new activity record must already exist preventing the creation of 'chains' of activity records from the invocation of one operation. With this new type of operation we could generate a structure that mirrored part of the activity structure on the domain. With this specification of the information system, there are a number of operations that are available to the information system that are nevertheless 'non-sensical'. In line with the discussions in this section, we should seek to minimise these non-sensical operations - unusually it is fairly straightforward in this case.

Firstly let us see what these non-sensical operations are. We can easily see one where a number of different *.Start* operations are presented to the user that do not correspond to operations that are permissible in the interaction theory. Suppose we had the following values of *dis-Activities*, *crs-Visit*Type, *dis-VisRel*, and the interpreted equivalents:



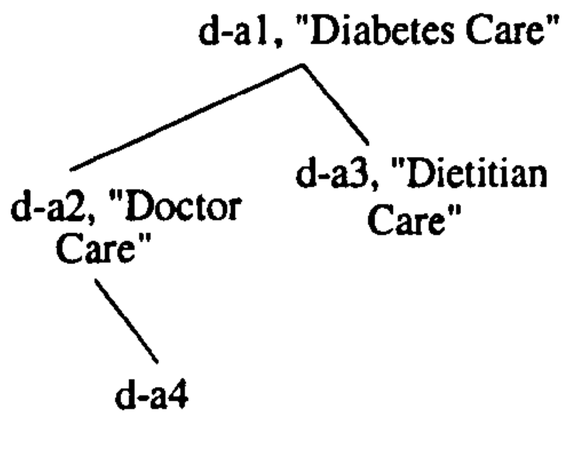
A graph showing values of *dis-Activities*, *crs-VisType*, and *dis-VisRel* for a particular model of the system described.



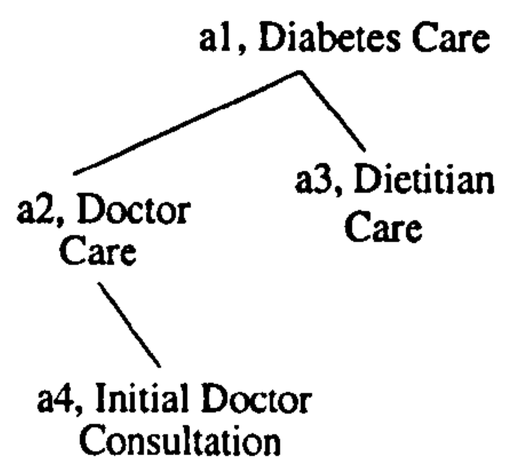
A graph showing the interpretation of those values on the left.

Figure 3-20: An information system model and its interpretation

Now if we want to generate a new activity as a request that was *During a1*, and record this on the information system we would have a choice of types to choose from. These types would be all possible 'child' types of Doctor Care that were supported by the information system. This is the set {Initial Doctor Consultation, Follow-up Doctor Consultation and Doctor Telephone Consultation}. Suppose we choose Initial Doctor Consultation. If we have created a request, the information system will not be able to record the associated type. The new state of both information system and domain, as given by our model of the interaction theory would be:



The graph of the state of the information system after the creation of the new request record.



The new state of the domain after the same (interaction theory) operation. Notice that although *a4* is of type Initial Doctor Consultation, we cannot know this through observing and interpreting the information system.

Figure 3-21: A new information system model and its interpretation

The activity record *d-a*4 is in the set *dis-Request*, and so is not associated with a type record. When we come to start the activity in the interaction theory, we move *a*4 to the set *Proceed* and *d-a*4 to the set *crs-Proceed*. In so doing, we must assign a type record to *d-a*4. The information system can not determine the 'correct' type record - ie "Initial Doctor Consultation" as it contains no record of the type of the activity it is in the domain. Thus the information system must allow one of at least three type records to be assigned to *d-a*4: "Initial Doctor Consultation", "Followup Doctor Consultation" and "Doctor Telephone Consultation". Two of these operations are invalid in that they lead to inconsistent states of the interaction theory. They are valid operations constrained by the interaction theory and are thus what we have described as non-sensical. We can show the three operations in the following diagram:

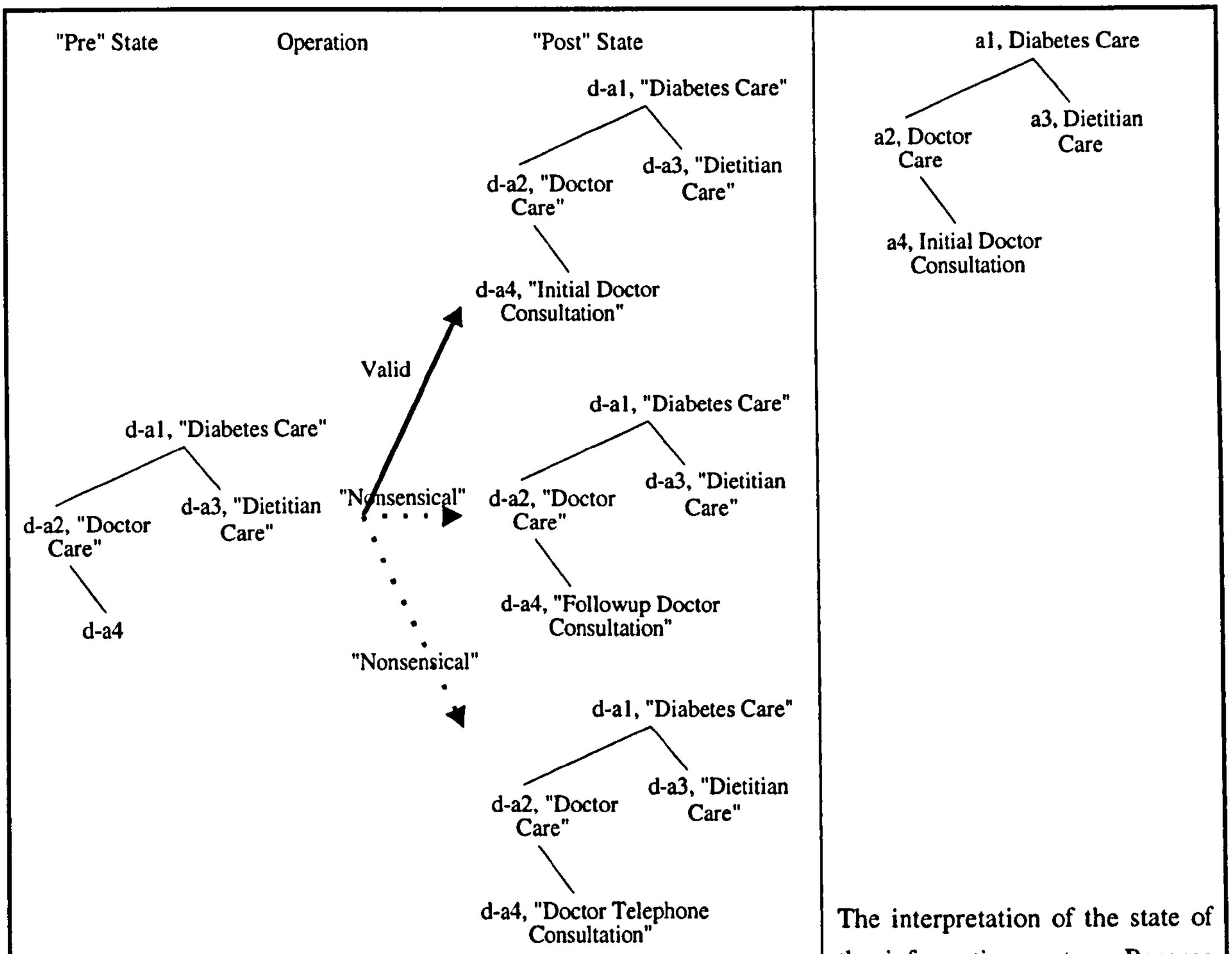


Diagram showing one valid and two 'non-sensical' operations that the information system will present to the user when she wants to record the start of activity *a*4. (In fact, starting *d-a*4 as a "Doctor Telephone Conversation" would be prevented as such an activity cannot be requested. This point is incidental to our argument however, and has been ignored.)

The interpretation of the state of the information system. Because the type of *a*4 does not change, two of the operations on the left are 'non-sensical'.

Figure 3-22: Non-sensical information system models and a permitted interpretation

This example illustrates well the increased 'entropy' of the system that the user has to overcome. There are other cases where the lack of a type record associated with members of *dis-Request* can lead to non-sensical operations. For example, given a member of *dis-Request*, the information system might allow us create a child activity record, given that one of the allowable types of the request record is in *Dom*(*crs-TypeParent*), whereas the interpreted activity in the domain is of a type that does not allow for child activities. It is not difficult to think of a number of other 'scenarios' that result in the possibility of non-

sensical operations. At least some of these problems can be addressed through the extension of *crs-Visit*Type to cover *dis-Request* as well as *crs-Visit*. This extension is a new state component - we have chosen to call it *dis-ActType* where

dis-T 6': *dis-ActType*: *dis-Activities* → *crs-Types*.

Now, with the introduction of this new state component, the creation of a member of *dis-Request* must be accompanied by its assignment to a member of *crs-Types*. Thus when the activity is requested in the domain, the relevant type is assigned in the information system. The information system presents no non-sensical operations to the user inasmuch as all the options correspond to valid operations in the domain. This is not to say that the domain operation will always be invoked correctly - a doctor could record that she was running a Doctor Telephone Consultation on the information system when she was in fact running a Followup Doctor Consultation. We cannot do anything about misuse of the system, at least not with the logical structure we are investigating here (this is more likely the role of training and good technical support). If we further reduce the number of information system operations here, we will be preventing the recording of valid domain operations.

It should be noted that the introduction of *dis-ActType* and the other attributes of *dis-Request* would also be motivated by the desire to expand the scope of representation of the system. As was explained earlier, one decision might have several motives. The decision in the case above is not quite as straightforward as the examples given in the sub-section covering expansion of scope which is why it has been included in this sub-section.

In all the examples given above we have reduced the entropy of the system by removing non-sensical behaviours. This helps guide the user by preventing them from making certain classes of error in behaviour invocation. We must be careful to make a distinction though, between the recording of non-sensical behaviours and the incorrect usage of the system. The essential difference between the presentation of operations that are non-sensical, and the invoking of operations that are incorrect representations of changes of state in the domain, is a logical one. In the case of the non-sensical operations described above, there is no operation that can be invoked on an appropriate model of the interaction theory that can lead to the changes of state of the information system. In the case of the incorrect usage of the information system, there is always a valid operation in the domain that can correspond to that invoked on the information system - in practice we cannot be sure that this valid operation is the correct one however.

One last point ought to be made that is pertinent to this part of the argument. It was stated above that all the operations presented to the user, in this small part of the information system specification, were now valid and represented possible changes in the state of the domain. This is still not entirely accurate - there are some operations that are supported but are non-sensical because they are disallowed to the class of user of the system. In the domain class TypeClass4 the structure called *EmbedType* prevented certain sorts of health care professional from generating certain types of activity. This particular constraint is not represented at all in the information system, meaning that in a DIS1 system that implemented the specification described here, a dietitian could cause an activity record of the type "Followup Doctor Consultation" to be created. This is a non-sensical operation as the domain theory prevents its interpretation from occurring. In order to prevent non-sensical operations of this class, an entirely new group of concepts and state components would have to be introduced to the information system, thus greatly complicating its structure. As has already been argued, all the developmental motives are to be tempered by the desire to maintain clarity and simplicity in the finished system: the benefits of

representational accuracy should be weighed against the costs associated with difficulty of implementation and use. In the case of introducing a representation of health care professionals into the system, we would gain little at the cost of a great deal of extra complexity, and so the 'improvement' should be rejected[xx]. In the case of the extension of *crs-Visit*Type to cover all members of *dis-Activity*, we reduce system entropy by a modest amount, but the increase in complexity is negligible: for this reason this improvement was introduced.

The next sub-section will investigate the converse of this motive, where the intended use of the information system does not place constraints on the possible behaviours exhibited by the information system, but rather on the behaviour of the domain. In other words the intended usage of the information system will prevent certain otherwise valid state changes in the organisation from taking place.

12.4.6 Minimisation of prohibition

The last 'motive' we come across is that which encourages us to minimise the restrictions that the information system places on the domain through the interaction theory. In many ways this is similar to the last motive: in the last section we saw how we should prevent the domain from constraining the information system, now we want to prevent the information system from constraining the domain. Whereas in the case where the constraint was placed on the information system we observed that that system presented the user with too many choices, in the latter case, where the information system constrains the domain, we say that the system presents the user with too few choices. Behaviours that are valid in the world are prevented through the use of the information system.

We might ask ourselves how this is so here: most of the interaction theory operations that we have seen only optionally invoke the relevant operation from the information system specification. Many of the invariants describing the interpretation of the information system state components are expressed as predicates involving the subset relation. The use of the subset symbol here generally means that a certain aspect of the domain is only partly represented in the information system. For example from the invariant

dis-int-I 22: $SupAct \cap Complete \setminus (im\ IntA)\ Cod(dis\text{-}VisRel) = (im\ IntA)\ crs\text{-}Complete$

we can deduce

$crs\text{-}Complete \subseteq (im\ RepA)\ Complete.$

which tells us that some of the activities in *Complete* are represented as members of *crs-Complete* in the CRS database. Indeed, the interpretation and representation functions themselves inform us of this partial representation. The type declaration

dis-int-T 5: $RepA: Activities \twoheadrightarrow dis\text{-}Activities$

says that only some activities are represented in the system: we have given this set the name *SupAct* through the invariant

dis-int-I 14': $SupAct = Dom(RepA).$

[xx] In this particular sub-domain. If the system were required to record the category and identity of its users for security and audit purposes this extra complexity associated with the introduction of 'Health Care Professionals' into the specification might very well prove to be worthwhile.

We know that there may be some activities that are not supported by the system because we present two operations in the interaction theory that can generate new activities in the domain and the information system: these are ***.NonRecGenerate*** and ***.RecGenerate***. The first operation causes the creation of an activity in the domain independently of the information system - an instance where the creation of an activity in the domain has not been recorded in the information system. The second operation describes the case where the creation of a new (subsidiary) activity has been recorded in the information system - both the state of the domain and the state of the system have been updated simultaneously. Notice that there is no 'third case' where the information system is updated independently of the state of the domain. We said earlier that we prohibited this as we wanted to talk about the behaviour of the information system when it was being used as intended. This property of the interaction theory is also enforced through the invariants: IntA is a total function:

dis-int-T 5: $IntA$: $dis\text{-}Activities \rightarrow Activities$

so we know that there are no activity records in the DIS1 system database that do not represent 'real' activities from the domain. We have thus placed a constraint on the behaviour of a model of the interaction theory that represents the use of the information system as we expect it to be used. We can go further with our impositions, and make more rules that prescribe how the system must be used, and investigate the implications of this declaration.

For example, we have insisted that once an activity is represented in the information system, its life-cycle is intimately involved with its representation. We can create an activity that is not represented in the information system, but if we choose to invoke the information system operation at the same time through the use of the operation ***.RecGenerate***, we cannot thereafter explicitly start the activity in the domain without recording the fact in the information system. This is because any activity record explicitly created through the ***.RecGenerate*** operation must be childless, and one intended use of the information system is that the start of all requests represented by a childless record in *dis-Activity* is recorded - this is specified through the invariant

dis-int-I 21: $SupAct \cap Proceed \setminus (im\ IntA)\ Cod(dis\text{-}VisRel) \subseteq (im\ IntA)\ crs\text{-}Proceed.$

This close coupling of the lifecycles is not truly realistic. As explained earlier, we have tried to limit the possible dissociation between the domain and information system in a fairly artificial way as this helps us in the construction of the interaction theory. This is not a problem as long as we recognise that, at the current level of sophistication, the interaction theory doesn't completely deal with the issue of the interpretation of the state of the information system lagging behind the state of the domain. We see this again reflected in the ***.End*** operation where if the activity being ended is supported in the information system, then the representation of the activity is 'ended' in the information system.

In the cases described above, domain operations must be accompanied by the relevant information system operations, but in none of them is an operation in the domain prevented. We can now see how we might describe the prevention of a particular operation however. If the system operation was constrained by harsher preconditions than that in the domain, and we had insisted that the latter could not proceed without the former, there might well be occasions where the behaviour of the domain was constrained. For example, suppose we said that an activity record could only be moved to the *crs-Complete* set if its type record was taken from a proper subset of *crs-Types*. If this were the case then there would be times when the activity record could not be so reassigned, and as we had insisted that the domain operation be accompanied by the system operation, the state of the domain would have to remain constant too. Of

course this is a ridiculous situation, and if we found this to be the case we would have to change the nature of the interaction theory.

There are times, however, when this sort of thing is not ridiculous. We might be forced into the position of relying on computerised support, or we might want to be able to use computerised support, but do not want a parallel manual system to be used. The example that will be discussed now, the definition of the booking system, is one of these.

Suppose we wanted to introduce the booking component of the DIS1 system into the Diabetes Day Centre. We know from our existing exploration of the interaction theory that there are certain types of activity that are not supported by the information system. There are some types whose activities can not be booked - those that can are in the set *Bookable*. Of those types that are *Bookable* and are supported by the information system - ie the members of the set $Bookable \cap SupType$ - we would expect that some are to be booked through the appointment system. We would probably require that all of the activities of some types that are booked should be booked through the computer - in other words, for some types computer aided booking is desired and should not be complemented by a parallel paper based system. We cannot in fact prevent a paper alternative being used in practice, but we can see what the effect of the hypothetical enforcement of such a policy would be on the running of the clinic. The way we do this is again to describe the intended use of the system through the interaction theory, and see what effect this has on the possible behaviours of the domain.

We have a number of types and invariants already in place that help us to do this. We have said that activities of certain types, or at least an identifiable subset of such activities, are represented fully by the DIS1 system. The set of types, as we have already seen, is called *FullRepTypes*, and the particular subset of activities of these types is defined by the invariant

dis-int-I 20: $(im\ ActType^{-1})\ FullRepTypes \cap Proceed \subseteq (im\ IntA)\ crs\text{-}Proceed.$

The set of types that can be booked by the appointment system is represented by the set *dis-Bookable*, and can thus be written as

$(im\ IntT)\ dis\text{-}Bookable$

where (as we can infer from the definition of *SupType* and invariant dis-int-I10)

$(im\ IntT)\ dis\text{-}Bookable \subseteq Bookable.$

Now those types that must be booked by the information system is clearly a subset of $(im\ IntT)$ *dis-Bookable*: in the interaction theory presented here, we have assumed that this subset is in fact

$(im\ IntT)\ dis\text{-}Bookable \cap FullRepTypes.$

This is a reasonable assumption, and saves us from having to introduce a new named subset of *Types* to describe this situation (the reader can probably think of occasions where this assumption is invalid, but as long as we are clear that we have made it, we can proceed safely).

One more assumption we have made which may not be always accurate but seems reasonable is that not only must all activities of the set described be booked through the computer, but no others may be: the computer provides full booking support for certain types of activities or none at all. This means that the set

(*im IntT*) *dis-Bookable* ∩ *FullRepTypes*

is equal to

(*im IntT*) *dis-Bookable*.

To summarise then, we have said that whenever an appointment is made for an activity of a type taken from the set (*im IntT*) *dis-Bookable* the appointment must be made through the computer system, and that only activities of this type may be booked using computer support.

The enforcement of this decision in the information system has some interesting implications. Let us consider the booking operation that books a previously non-existent activity which we have called ***DIS1Interaction.Book***. This operation always invokes the domain equivalent: ***ATClass5.Book***. We have said that whenever the type of the activity to be created is not of a type taken *from* (*im IntT*) *dis-Bookable* then the information system operation is not invoked. For those types that are, then we can only invoke the interaction theory operation if we also invoke the information system operation. We thus have a skeleton operation schema that looks like this

DIS1Interaction.Book(...*t*...)

[Type Declarations]

ATClass5.Book(...*t*...)

. 1 dis-int-Pr 19: $t \notin$ (*im IntT*) *dis-Bookable*

. 2 dis-int-Pr 20: $t \in$ (*im IntT*) *dis-Bookable*

[Precondition Case 2]

DIS1Class1.Book(...*RepT*(*t*)...)

[Postcondition Case 2]

It is the contents of the precondition labelled [Precondition Case 2] that we are interested in. If we look at the operation schema in the interaction theory in Appendix 5 we see that there are many complex preconditions. Most of these relate clinic lists to their representations (the logic and structure of domain clinic lists is different from that of their representation), and do not constrain behaviours of the domain. The precondition:

dis-int-Pr 26: $\exists sl$: *dis-Slots* • *SlotStart*(*sl*) = *RepTime*(τ_b) ∧ *SlotEnd*(*sl*) = *RepTime*(τ_e) ∧ *TypeLink*(*RepT*(*t*)) = ((*as-ClinicType* ° *as-StreamClinic* ° *as-SlotStream*) ◊ *as-SlotDay*) (*sl*)

does constrain the domain fairly harshly. What the precondition says is that the operation can be invoked only if the times of the appointment are such that there is an appropriate slot record (ie of the correct 'slot type') in the OPAS that has commensurate start and finish times (as the number and length of slot records is decided in advance via the information system operation ***DIS1Interaction.SlotsCreate***). We saw in Section 12.2 that the OPAS assigns a discrete number of slots of defined length to a particular 'stream'. The duration that the slots can take depends on the 'type' of the clinic, and is fairly tightly constrained. Given the way the appointment system would be used, the slots for a particular clinic would be assigned

and defined weeks if not months in advance. The domain theory that has been presented has none of these restrictions. The clinician is free to create a (domain) slot of any length when she makes the booking. In the interaction theory then, if we insist on the use of the computer to book certain types of activity, there are many behaviours that are prevented.

An example of this can be taken from the DEDC. Once a week, a 'new patient' clinic is held at which all the attendees are precisely that. On other days, a mix of new and followup patients will be seen. New and followup doctor consultations are represented by different 'slot types' in the OPAS as they take different lengths of time. The appointment system forces the decision about how many new and followup patients can be seen on a particular day to be made (probably well) in advance of the booking being taken. Thus when a patient is booked in for an appointment in a few months time, he or she must be allocated to a pre-existing follow up patient slot record. If all of these have been allocated for a particular day, then another day must be chosen, even if there are vacant new patient slots for that day. Another possible domain behaviour that would be prevented by the interaction theory is the allocation of times for appointments according to assessed patient need. If a policy of this type were being used, then a well controlled diabetic patient could be booked in to see a doctor for 10 minutes whereas one with complications might be booked in for 40 minutes: booking regimes run like this have been shown to be more efficient in their use of resources. This domain behaviour would likewise be constrained if the use of the OPAS system to book such activities were enforced.

We have discovered a particular and well defined area where the introduction of the information system into the domain constrains the behaviour of that domain. We can interpret this finding and react to it in a number of ways, not all of which reflect adversely on the information system.

Firstly, the specification of the OPAS is not the same as its implementation. The theory of the OPAS that has been presented was derived from a cursory 'reverse engineering' of the proposed system (it is not implemented at the time of writing this thesis). This method of specification derivation is always difficult, and it is possible, or even probable, that the presented formal theory has many errors of representation. The understanding of the system might be erroneous in just this area of slot creation and appointment booking. Before the OPAS is criticised for imposing unreasonable constraints on the running of the clinic, we should be sure that our understanding of its behaviour is accurate.

Secondly, the information system might be a more precise representation of this part of the organisation than the domain theory. The logical mechanism we used for refuting, and thus enhancing, the domain theory was that of counter-example discovery. A theorem derivable from the domain theory was considered 'correct' until an example of a behaviour prohibited by the theory was observed, or claimed by an interviewee. If the domain theory is correct, all the behaviours it prohibits will never be seen in the organisation: it might, however, permit behaviours that we never see in the organisation. If this is the case, we should not say that the theory is wrong, but rather that it could be made to be 'bolder' in its prohibitions and still be correct. We have seen that the information system as described in the interaction theory prevents some operations that are valid in the organisation: these excluded operations might never be observed in the domain. In short, both the domain theory and the information system specification might be equally accurate theories of a generic clinical grouping in this area, but as the DIS1 specification is bolder, then it can be thought of as better. To determine which theory (the domain theory as it is, or one that permitted the same behaviours as the OPAS system) was better in this case, we would have to investigate those behaviours allowed by the current domain theory and prohibited by the information system (when interpreted as a domain theory rather than an information system specification). If any behaviours in this set were observed or claimed to exist by a stakeholder, then the OPAS specification

would genuinely prevent otherwise valid domain operations. If such a behaviour was never observed or claimed, then we might reasonably take the domain theory as embodied in the OPAS as a better one than that which has been presented in this thesis.

We should not be surprised if some behavioural areas of the organisation were better represented by the information system than by the domain theory that has been presented. Not only is it likely that many more man years have been expended on developing the OPAS than the theory, but those associated with the latter have been concentrated on supporting a much smaller domain than the theory presented in this thesis.

Even if the behavioural description embodied by the OPAS were inaccurate, we still have to decide whether or not we are willing to live with it. It might be that the behaviour described by the OPAS specification is significantly easier to implement than that described by the domain theory. Certain structures and behaviours are susceptible to ready representation by the relational model while others are more suited to representation through the use of matrices and tensors (which is the underlying logic of the APL language). In these cases, there is a trade off to be made between the benefit of supporting the operations in the domain that would otherwise be prevented by the introduction of the integrated DIS1 system as described and the cost of designing the OPAS to more closely mimic the domain: these costs and benefits must be assessed and an appropriately informed decision consequently taken.

Of course, it might be that the OPAS is a poor model of the organisation in precisely the manner described, and it might be that the constraints placed on the clinic were it to be introduced would be absolutely intolerable. Which explanation for the difference between the domain theory and information system specification is most realistic has yet to be discovered: for this reason, no attempt has been made to alter specification of the DIS1 system accordingly (the prohibited behaviours could be re-introduced with the help of suitable extra state components defined at the integration stage - ie *dis*-... variables). What can be said is that the use of the interaction theory here has illuminated an area that might have serious repercussions on the usability of the implemented system. The decisions taken as a result can now be informed rather than blind.

One last thing that ought to be said in this section is that the constraint that the OPAS will place on the clinic should not be seen as a conscious policy change. Although we are assessing the validity of the information system in terms of its interpretation onto the domain theory, what the specification 'really' represents is a series of data sets and algorithms: any policy change should be described through an extension to the domain theory. Behaviours of aspects of the organisation are represented through models of the domain theory. Of course, once the domain theory has been modified, we can see how successful a particular information system design will be in supporting the new procedures and policies. Although the exploration of such 'hypothetical' domains is not the subject of this thesis, an example of such a domain theory is included in Appendix 6 investigating possible constraints needed to implement the 'contracts for service' being encouraged in the NHS.

12.5 Conclusion

We saw in this chapter how an interaction theory might be used to motivate decisions taken in the design of new information systems. The specification of an information system that is to be implemented at St Thomas' Hospital, the Out-patient Appointment System (OPAS) was described, as was the specification of an information system that is an integration of this and the earlier described CRS. This is the first fragment of a Directorate Information System and so was called DIS1. The main purpose of this chapter, however, was to show how an interaction theory might be of use in the design of an information system.

To this end a number of ways in which the construction of the specification of DIS1 were influenced by the interaction theory were presented. There are two conflicting considerations when designing an information system. The first is the need for simplicity of design to facilitate construction, maintenance and use of the system. The second is the desire for the system to say as much about the world as possible - in short for it to 'conform' to the domain theory. Four ways in which such conformity should be sought are described. These are called the four developmental motives as they provide the motivation for design decisions which concern increasing the representational verisimilitude of the computer system. These four motives encourage us to seek

- the gratuitous expansion of scope,
- the functionality of interpretation,
- the reduction of entropy, and
- the restriction of prohibition.

One conclusion of this chapter is that the role of formalism in the understanding of the use of information systems is small. All the motives discuss the 'intended' or 'correct' usage of the information system being considered. They have not said anything about how we can ensure such correct usage. There are a number of ways in which we can achieve this (apart from being an improved representation of the domain) such as good user interface design, user training and management (although a statement of the correct usage of the system such as that given might well help with the running of a training scheme).

That there are many aspects of information system design that the process of formalisation offers no support for at all is not a problem - the methods that are advocated in this thesis do not purport to be the 'answer' to the problems of information system design, they are merely of some use in understanding aspects of the highly complex environment in which the analyst works. In the case of this example, we have not arrived at the best information system to support the booking process: we have merely taken an example of a possible design, shown that with a particular interpretation this design led to inconsistencies of representation, and presented a second design which, with a similar interpretation, avoided these inconsistencies.

It must be remembered that the interaction theory is only one of many possible interpretations. What we have done is made an assumption about how the information system will be interpreted and used, and 'improved' the system on the basis of that assumption. The testing of a given interaction theory is outside the scope of the thesis, but would be an interesting area for further work. This point is considered further in Section 13.4.

Another point that is apparent is the way in which arbitrary choices in the construction of the domain theory significantly influence the nature of the interaction theory. For example, the representation of the cancellation of an activity as moving it from *Proceed* to *Request* reduces the clarity of the interaction theory - certain invariants must be understood as applying to requests that were once proceeding activities, although this cannot be described in the static schemas. Similarly, the representation of time in the domain theory is quite different from that in the OPAS. As a result, the interaction theory is rendered more complex without any corresponding increase in insight. The wisdom of altering the domain theory in the light of the information system specification is uncertain however. The construction of the domain theory should be influenced more by the experiences of the domain workers, than by the information systems that are to be developed to help them.

One final point that will be made here concerns the way in which the interaction theory might improve the domain theory. An early version of the specification of the CRS assumed that all 'childless' visit records were of a type where the patient had to be present before the activity being represented could be started (this is so far the case - support for telephone conversations and other consultations where the patient need not be present is not provided: this was not considered sufficiently important to keep in the specification however as it represents an accidental, rather than designed, property of the system). While this was the case, we might have expected to see an invariant of the type

$$(im\ IntV)\ ((im\ VisitType^{-1})\ (is\text{-}Types \backslash Cod(crs\text{-}TypeParent)) \cap crs\text{-}Proceed) \subseteq ActAtt$$

to be in the interaction theory. That is, all visit records in the set *crs-Proceed* that are childless should be interpreted as activities that the patient is currently attending. When questioned whether an invariant of this type should be in the interaction theory, the system manager of the department's current CRS said that it should not be. The reason for this is that there is a problem with the domain theory concerning activities that the patient did not attend but were completed anyway. These activities are called DNAs in the clinic, and are an important aspect of its functioning. As a result of DNAs certain tasks are engaged in, and during the DNA the clinician might well look at the patient's notes and add to them. This is an example where the specification of an information system and the construction of the interaction theory can help to refute the domain theory thus leading to its improvement.

Part Four:

Review and Conclusion

Chapter 13: Review of Results

It will be argued later that the hypothesis of this thesis has been shown to (more or less) hold. However, the issues arising from the work need more careful consideration if we are to benefit from the method: the project will only make a real contribution if others can learn from the problems encountered and mistakes made by the author. For this reason, in this chapter, we will review the lessons learned from the work that has been carried out over the past three years. Firstly the basic assumption underlying the project is described, the method that was used and why that method is beneficial if one accepts the underlying assumption. The remainder of the chapter goes on to question various aspects of the method in the light of the experience gained over the course of the project. Firstly the underlying assumption is questioned, and then each of the three major steps in the method. Each of these sections is laid out as a succession of 'justification, criticism, synthesis' subsections. The justifications of each part of the argument are presented, followed by a discussion of its major flaws. The virtues and vices of that part of the method's argument are consolidated in the synthesis. This is one of the main theoretical contributions of this thesis. The method has been carefully thought out and explained, and then reinterpreted in the light of significant experience of its use. Although the philosophical background to the method is not greatly different from others that have been proposed (for example Checkland's Soft System Methodology), the extent and nature of the use of formalism (especially concerning the interaction theory) is perhaps unusual. The lessons learned from such use (and the goals sought through such use) are presented in this chapter. The insights gained through the extended 'case study' both as presented here and as gleaned by the reader from the results, are intended to contribute to the discipline known as 'computer science'.

The way in which this chapter will be presented is as follows. Firstly the method and its rationale will be briefly reiterated. Each major component of the method will then be examined in turn. Each examination will have three parts: firstly the justification for this aspect of the method is presented, then the limitations and drawbacks are considered as criticisms, and finally a synthesis of the two arguments is presented which describes how to view this part of the method.

13.1 Synopsis of method and rationale

13.1.1 An Assumption Underlying the Method

There is a basic assumption underlying the work reported in this thesis. We must elucidate this if we are to understand why the method is as it is. The assumption is as follows:

That users of information systems interpret (some) of the state and behaviour of the information system into objects in the 'real world' as they perceive it, in particular that part of that world related to the task being supported by the computer system (called the domain in this discussion).

It is further considered that difficulties of interpretation are manifested when the state components of the information system do not behave isomorphically with the aspects of the domain that those components are imagined to represent. These difficulties of interpretation are a significant cause of user dissatisfaction with the information system. Furthermore a systems development method that in some way facilitates the design of an information system that can be robustly and reliably interpreted into the domain by its users will result in more useful systems.

Although this assumption is partially justified above (in Section 4.3) and discussed further below (in section 13.2.2) it is not part of the argument presented in this thesis - rather it is a postulate on which the

argument is based. Having stated it we are in a position to consider the method in its entirety prior to investigating its component parts.

13.1.2 Method

As a result of the underlying assumptions that have guided the analysis presented in the thesis, a particular method has been used. One can never determine directly how a user is interpreting an information system when it is being used - to imagine that we can do so when the information system is proposed as opposed to already existing is doubly far-fetched. In order to prejudge this interpretation, we must approach the problem indirectly.

The method used in the project works through constructing just such an indirect path to the understanding of system interpretation. The analyst cannot access the interpretation of the information system directly, but can judge the reliability of the interpretation through inspection of this indirect path which represents a design process. The design process consists of three steps. Starting at the domain, or rather the perception of the domain by the users, step 1 is the construction a theory of the domain. Step 2 is the derivation of the theory of an information system, more commonly known as a system specification, and step 3 is the implementation of the specification as an operational computer system. If this is done 'correctly', the resulting computer system should be capable of being interpreted back into the domain by the users in an intuitive manner. This process is illustrated by the diagram below:

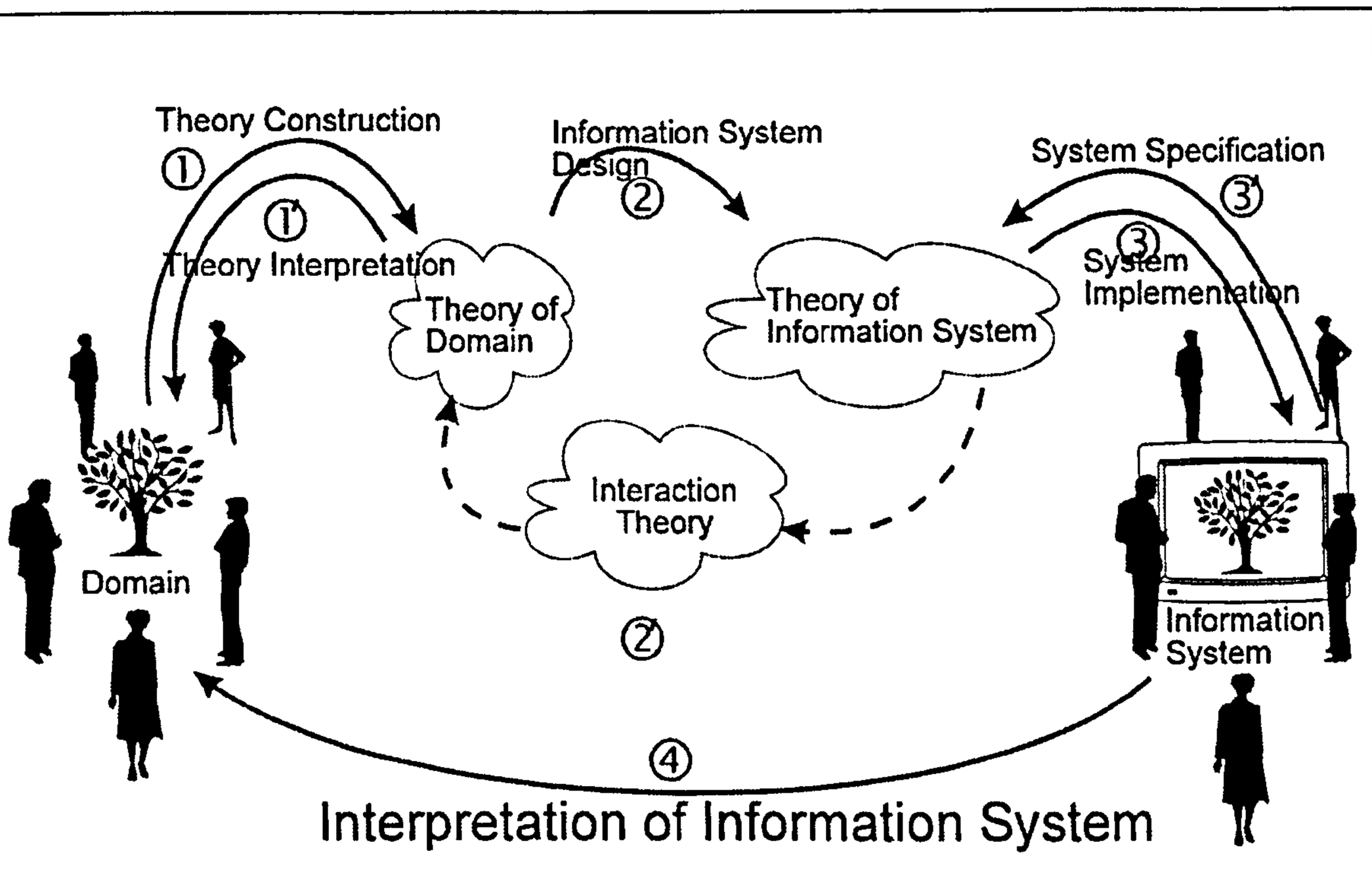


Figure 4-1: The systems design process

Diagram illustrating method used in thesis. Stages 1, 2, 3 and 4 are commutative. We can interpret an information system into the domain either directly (via step 4), or indirectly (via steps 3',2', and 1' - the reverse of steps 3,2,and 1). The analyst uses steps 1, 2 and 3 to produce the information system. The users use step 4 to interpret the information system into the world. If such interpretation is not possible, it is because an error has been made in one of steps 1, 2 or 3.

The word 'correctly' was used in inverted commas because we can never carry this process out completely and perfectly, not least because we are attempting to describe an (essentially) infinite world using finite

language. The information system will thus only be a partial representation of the domain. Moreover, we might have to make compromises for the sake of ease of implementation, speed of use, tractability of computations and so on. All these compromises will have an effect on the usability of the system in terms of its interpretation (though a perfectly interpretable information system might be unusable for other reasons, for example extreme slowness). The method separates these compromises from the construction of a theory of the domain. If we carry out steps 1 and 3 correctly, then all the problems and inadequacies of interpretation should be apparent in step 2 - the derivation of the information system specification from the domain theory. By inspecting the workings of this step alone, we should be able to assess validity of interpretation of the eventual information system. The appropriate artefact that is significant at this stage is something that has been called the interaction theory in this thesis.

In order to confine discussions of interpretation, and thus according to the underlying assumption, usability, to the interaction theory we must be sure that steps one and three are correctly completed. Each of these steps is fraught with difficulty. Three of the next four sections discuss these problems and suggest a more realistic way of viewing the method. First of all we must inspect the underlying assumption and assess its validity.

13.2 Issues concerning Underlying Assumption

As with the other sections in this chapter, we will explore the issues concerning our underlying assumption through the use of a form of dialectic. The first sub-section maintains and argues that computers are used as models of the world. The criticism sub-section points out flaws in this interpretation and says that we should understand computers in a different way. The synthesis recognises that this new understanding is superior in many ways but within it we can see that computers can 'deceive' users into thinking they are looking at a certain sort of representation of the world and then proceed to let them down when this transpires not to be the case. This is undesirable and is one of the causes of user dissatisfaction with information systems.

13.2.1 Justification

The assumption that an information system is interpreted into the world by its users is so ingrained in the minds of many whose job it is to design and construct such entities that it is rarely questioned or even articulated. Referring to standard texts on systems analysis (such as [Avison88], [Downs92], and [Coad91]) we can infer that this is considered to be a property of the use of information systems from observing the importance such works place on understanding and representing the problem domain that the system will address. However, we should not have to rely on custom and habit for justification of the statement: indeed, by considering what a computer system is and how it will be used we can see that the assumption is eminently reasonable.

A computer can be most generally thought of as a device for manipulating symbols. It can create, destroy, shuffle, recieve and display (or, more generally output as the 'display' might not be visible) these symbols according to set of (symbolic) rules. What makes such machines useful is the way in which people are able to interpret those symbols as representing things that are of interest to them. Without such interpretation, computers would be mathematical curiosities - with it they are playing an increasingly major role in human society. The things that people are interested in might not be aspects of the 'real world': a word processor for example does not represent any part of the world, but its users still interpret the symbols it manipulates as things that are of interest to them - namely letters, words, punctuation marks, paragraphs and other aspects of documents. An information system is a particular type of

computer system, however, and the way in which it is used carries implications about the way in which the symbols it manipulates and displays are interpreted.

As we saw in Chapter 4, we can divide the workers in an organisation into two groups: the operational and the managerial. Both types of workers need to respond to changes in the organisation that have been brought about by other workers or by interaction with its environment. In order to do this, each worker needs to know what is happening in the organisation, and what has gone before. In other words she needs to know what the 'state' of the organisation is. It is the role of an information system to record this state, be it in on pieces of paper or in a computer's database. The computer system we are designing must thus act as a record of the state of the organisation - the only way it can do this is if the symbols it stores, manipulates and displays are understood as representing aspects of the organisation. In other words, in use, the users of an information system interpret it into parts of the organisation, or domain.

It should be noted in addition that the type of information system being designed is operational rather than managerial. In this respect it must record and present data which on interpretation describes the state of the organisation so that workers can judge what actions are appropriate to take. We are not interested in further massaging of that data to produce summaries, help predict trends or in any way help the manager deal with the profusion of data available from it: this would be the role of a management information system or, in the clinical case, a system to support epidemiology. Operational workers want to understand the state of the world as it is, not as it might be given a different sales forecast, or patient case-mix.

13.2.2 Criticism

The somewhat naïve view presented above has come under increasing criticism in recent years. Some of the criticisms (such as the subjectivity of the 'real world' and the impossibility of representing it in symbolic form) are raised later in this chapter and so will not be discussed here. Instead we will look at two ideas described by two authors in an influential book typical of the attacks on this 'classical' view of information systems: Understanding Computers and Cognition by Winograd and Flores [Winog87]. The first idea concerns the impossibility of neutral description, and the second the commitment that a good system designer must show to the user.

The first idea has much in common with developments in language theory in the 1930s, notably the development of speech act theory by Austin as first propounded in his book 'How to do things with words' [Austin62]. Earlier in this century a school of thought was pre-eminent in philosophical circles. The doctrine that this school subscribed to was that of logical positivism. This doctrine held that all meaningful utterances described some aspect of the state of the world, and as such it could be judged a truthful description or an untruthful one. Any utterance that could not be deemed to be true or untrue was simply rejected as meaningless. Austin showed that there were utterances that were definitely not meaningless and yet could not be said to be true or false. For example the phrases 'I arrest you in the name of the law', or 'I declare war on San Marino' do not describe the world - they act on it to alter it (assuming the first is said by a policeman and the second by a national ruler). Austin went on to show that the use of an utterance to act on the world is not limited to these formal and obvious examples, but applies to all uses of spoken language. Whenever we speak we do so for a purpose - to explain a point, to describe a situation, to start an argument, to defuse a row, to cause merriment and so on. In short, we all use words not just to describe the world, but rather to do things to it and our relationship with it. Indeed no utterance can be totally descriptive and achieve nothing, and in describing the world we are in fact also changing it: in the case of a description these changes are usually limited to an altered understanding of some aspect

of the world on the part of the listener. The changes associated with such a description can be infinitesimal (such as might be the case with the description of systems analysis techniques to a room full of bored students) or large (such as might be the case with the description of conditions in a famine zone on national television), but it is change of some sort that is the purpose of the utterance.

We can, with some care, extend the argument originated by Austin covering spoken utterances to other forms of communication, both for private means such as might be the case with a written letter, or for a public consumption such as might be the case with a novel or newspaper article. We could extend the idea still further to cover more diverse media such as painting, sculpture, music, and so on. In fact any time a medium is used to convey any form of message from the creator to an audience is an instance of communication, and as such should be seen as being motivated and non-neutral. Winograd and Flores concern themselves with cases where the medium of communication is a computer, especially information systems. There are two ways in which computers act as conduits for communication - the first is for communication between users, the second from the designer to the user.

In the same way that the spoken description is intended to change the world, so is the description contained within an information system. The users of an information system are endeavouring to effect change on the organisation they are describing. The motivation for recording data on a computer system might be simple (such as when the act of recording is specified in the user's job description) or complex (such as when a doctor records aspects of a patient's condition in order that she can rapidly assimilate the salient characteristics of his state of health at the next consultation). Although it is reasonable to suppose that the desired change in the state of the organisation is intended to be beneficial, it is certainly not neutral. We should bear this in mind when considering the design of the system - its users will not think of it as a neutral device for recording the state of the organisation, but rather a tool which can be used to change the world in some way and this use must be understood by designers if it is to be successfully introduced into the organisation.

A computer system is not a neutral object in its operation: neither should it be thought of as neutral in its design. For an artefact to be designed, there must be a designer or team of designers. In the case of an information system this designer (or team) with all her prejudices and preconceptions is often ignored, and the computer system is seen in itself as an impartial object. Although the object might be impartial, its actual or intended effect is certainly not. Winograd and Flores liken a computer system to a text which is used to communicate between a writer and a reader. Although the medium of the message is much more sophisticated than, say, the printed word, it is still a form of communication between, in this case, the designer and user. The ideas communicated are often potentially more insidious by virtue of the fact that computer systems are generally viewed as objective entities by their users. Indeed, generally neither the designer nor the users is aware of the potential for computers to convey messages from the former to the latter: this does not prevent the phenomenon - it merely renders it uncontrolled.

The recognition that computers are forms of communication from designer to user is an identification of a problem, not a solution: how should designers react to this revelation? Winograd and Flores suggest that any form of communication carries with it some form of commitment. For communication of any sort to work, there must be shared and understood commitments between the speaker and the listener. There is a commitment from the speaker to the listener to use words so as to imply a meaning from the utterance that the listener will reasonably glean. Similarly there is a commitment from the listener to the speaker to understand the utterance in the sense in which it was intended. For these commitments to work, the speaker must have an idea of the way in which the utterance will be understood by the listener, and the listener must be able to determine what a speaker must have meant from an utterance - these depend

wholly on the context in which the utterance was made, and the unarticulated concepts underlying the listener's and speaker's understanding of that context. An example given by Winograd and Flores illustrates the point

' ... it is impossible to establish a context-independent basis for circumscribing the literal use of a term even as seemingly simple as 'water' as shown by the following dialogue

A: Is there any water in the refrigerator?
B: Yes.
A: Where? I don't see it.
B: In the cells of the eggplant.'

In asking the question, A assumed there was a commitment between her and the listener such that B would understand that she was asking for water to drink (of course, one can think of contexts in which the answer given did not break any of the unspoken commitments likely to exist between A and B). If we are considering the design of a computer system as a form of message from the system designer to its users, we must understand what commitments exist if the communication is to work effectively.

In thinking about a computer system as a medium for various forms of communication, we have challenged the notion presented in the sub-section on justification presented above. Firstly users do not think of an information system as purely a representation of the world but as a tool to influence it, and secondly that a computer system is not a neutral object but carries a form of message from the designer to the users. That we can understand computer systems with a greater degree of sophistication does not mean that the earlier and simpler assumption is invalid: indeed, we shall see that this more subtle and insightful view renders greater still the importance of ensuring a good representation of the domain.

13.2.3 Synthesis

Although the above arguments question the simplistic reasons for arriving at the central assumption, our more sophisticated view need not change our conviction that we should build a computer system that in use can be interpreted into the world. We saw that designers as communicators are entering into a commitment with the listener, or user of the system. To comprehend what these commitments are we need to understand how information systems are used. We saw that the earlier assumption that a computer system is a neutral representation of the world is not wholly valid, and that a more accurate understanding has it that computers are used in order to create a change in the world. Workers change the world through communicating statements to each other concerning the domain they are working in. The terms that are communicated refer to things that are of common concern to the workers. However those terms are embodied, be they verbal, written or typed, there is a commitment on the part of the originator and the receiver that they should be understood as talking about those entities that are of importance to the shared task, and that that understanding should take place within the context of the domain of activity.

If we can see the way in which the users will exploit the system, we as designers can discern what (one of) our commitments as communicators must be. As users want to talk about aspects of the world of interest to them within the context of the domain of activity, we must provide them with a medium by which they can do this. One of the forms that such a medium might take is a structured representation of that domain. The fact that the system is structured means that less work is needed to construct a message, and the scope for 'error' is diminished. The fact that it is a reasonable representation of the domain of activity means that the terms that go into the creation of a message can be reliably understood as referring to the

entities of concern to the user. Of course there are many other ways in which we could construct a system that enabled users to communicate other than to provide them with a dynamic representation of their domain, but this is one way in which we can create a construct that fulfils our commitment - namely to allow the users to pass commitment making messages to each other thus influencing the state of the organisation.

We can see then that while the statement about information systems representing the world is not the whole truth it can be construed as an aspect of the truth. To this end the method described in the thesis helps. In some ways, and at some times, the user will expect the information system to behave as a model of the world - in providing an information system we are entering into a commitment to support this expectation. When we fail, the user will be dissatisfied. Of course any user of the computer system will have many other expectations - by addressing one we do not exclude the possibility that she will be disappointed elsewhere. We can say that a method based on this underlying assumption will help the process of systems analysis and design - it will not be a tool to cover all aspects of the task. However, as systems analysis is such a difficult process, we should jump at the chance to use any tool that will provide us with genuine help.

13.3 Issues concerning Construction of Domain Theory

In this section we will discuss the first step in the systems analysis / design process that was presented in Section 13.1.2. This step is the construction of the domain theory. Again the dialectic approach is used to help us in our exploration of the issues involved. The justification explains how we are constructing not a model of the world, but a theory of a small part of it called variously the universe of discourse, the domain of discourse, or just the domain. The similarity of the approach with the scientific method is explained and used (with reference to the latter's 'success') as a justification. The criticism questions this faith in such a 'rationalist' approach and presents objections which are themselves justified by calling on philosophical works and the experience of the project. The synthesis accepts that the problems associated with a more relativist and deconstructionist view of reality will not go away and are indeed fundamental to our way of thinking, but that if we accept the need for consciously created artefacts such as a computer system, we will inevitably be forced to make the philosophical 'mistakes' exposed by such a discussion. The approach advocated does not address these deep problems that will always occur, but avoids others that many methods make in addition.

13.3.1 Justification

The first step in the information system design process used in the project is the construction of a formal theory of the domain of discourse. The thesis, or argument, that we are discussing here is that we can understand the world, and specifically that we can understand it through the use of formal theories that have been exposed to 'experimental' refutation.

We are interested in understanding a part of the world as it is perceived by a group of workers, in our case the staff of the Diabetes and Endocrinology Day Centre and to a lesser extent of other directorates. We call the part of the world (construed by the workers) that we are interested in the domain. We do not want to make a representation, or model, of the domain - rather we want to understand it and its characteristics. To this end we construct a theory which can be used to judge the verisimilitude of any models that might be made. If our theory is correct, then the real organisation (or domain) that we are analysing should be one of its possible models. If this is the case then other models, if interpreted into that organisation, might exhibit behaviours that, when interpreted, will be observable in the organisation. In

this way, we are not describing the domain we have chosen to analyse, but rather a class of objects all of which share some behavioural similarities with that domain.

As described, what we are doing has many parallels with the scientific method elucidated by the philosopher Sir Karl Popper in his book 'The Logic of Scientific Discovery' which has been extensively discussed elsewhere. In this work, Popper points out that we can never prove a theory, but rather only disprove, or refute, it. A scientific theory attempts to explain some aspect of the world through the positing of behavioural rules that that aspect of reality obeys. If reality is observed to behave in a way that has been forbidden by the rules in the theory, then the theory is in error and should be discarded: we say that it has been refuted. There are two points we should bear in mind that apply to scientific theories as well as those that endeavour to describe social entities such as organisations. These are that it is in general impossible to prove a theory, and that an unrefuted theory might allow behaviours that are never observed.

The Impossibility of Proof

However, we cannot assume that the theory is correct, just because we have not observed behaviours that refute it.

Firstly we might not have found any counter-examples because we did not look. Popper explains in his autobiography [Popper92] that the event in his life that caused him to think carefully about epistemological issues and thus develop the scientific method was his flirtation with Marxism and socialism. He argues that for a while in his youth, not only did he believe that the end justified the means, but also that the means would inevitably lead to the end. It was only after a violent demonstration in Vienna that the young man reflected on what he believed and realised that not only was there no evidence to suggest that the socialist utopia would be realised through the means advocated by Austrian Marxists, but that there was significant reason to suppose that the Marxist theory was wrong. Predictions that Marx made had been refuted, and the social change he forecast was nowhere to be seen. A theory of a clinical department is a small thing compared to one that purports to describe social progress (and influences it considerably). Nevertheless in both cases the need for intellectual honesty remains - we must try our utmost to disprove any theory before we place any faith in it.

Similarly, we might have looked for counter-examples but not found them. Sir Isaac Newton's laws of mechanics were tested in extreme conditions for many years and were observed to hold from the very small (gas molecules) to the very large (the movement of planets). It was only with the advent of instruments and techniques that revealed the behaviour of components of reality that were even smaller (elementary particles such as electrons), even bigger (the universe) or faster (light 'particles') that it was revealed that the theory was at best only an approximation of those that were introduced subsequently (quantum mechanics, the general theory of relativity, and the special theory of relativity), and at worst plain wrong.

Neither of the 'erroneous' theories described above could be claimed to be correct, but we would be justified in saying that Newtonian mechanics was corroborated to a greater degree than Aristotle's theory about women's teeth as it was tested in more extreme circumstances. The reason why we cannot 'prove' a theory about the world is that we cannot observe (interpretations of) all its possible behaviours. This is at least as true about a theory of an organisation as it is about a theory about the physical world.

In fact this problem is more extreme when it comes to observing behaviours in a social domain such as an organisation. In the case of the physical world, we can test our theories with great rigour by controlling

and manipulating the environment of the 'domain of discourse'. This is the purpose of the 'controlled experiment'. We can rarely conduct such controlled experiments in an organisation - firstly because to do so would be unethical, and secondly because people are fundamentally uncontrollable in this sense. Our ability to seek counter-examples is thus severely curtailed, but this should not prevent us from looking for them. An alternative to the running of controlled experiments is presented later on in this section. Before we discuss this, we should understand another limitation of the process - that a theory might be correct and yet not say anything useful, or at least be less useful than another equally accurate one describing the same domain.

The Existence of Better Theories

The mechanism for determining the verisimilitude of a theory described above has been used for many years to a greater or lesser extent in the scientific community and has contributed to the faith that our society has in that community's findings[xxi]. The method that was used in the project to construct the theory has many parallels with this scientific approach, though with modifications to the experimentation process which are discussed below, and to the underlying philosophy (this is more modest than might be thought - although the author has a less classical and rationalistic stance than many scien ists, Popper recognised some of the flaws inherent in the method and accounted for them in his philosophy).

A mechanism for discovering errors in a theory is only half of what we need, however, and only half of the scientific method as described by Popper. The process of refutation tells us only whether or not a theory is correct, or more accurately whether or not a theory has been refuted. In the language of set theory the mechanism allows us to choose to which of two exclusive sets a (consistent) theory belongs - the set of theories that have been refuted, and the set of theories that have not been refuted yet. If we take the domain of electromagnetism, we will find many members of the set of not-yet-refuted theories. One of these will be Maxwell's theory. Another will be the theory that states 'At any time there are a number of electromagnetic waves each of which has an associated frequency' and ~~no more~~. While the former has been used to guide the development of the electronics industry ever since its inception, the latter tells us so little as to be almost useless - yet they are both members of the not-yet-refuted set. We can see that we need more than the logical division of theories provided by the refutation mechanism: we need a mechanism for discussing the quality of a not-yet-refuted theory so that we can order our corroborated set. Popper describes this qualitative aspect of a theory as falsifiability: in this thesis, it has been called its boldness which the author feels conveys the spirit of the idea.

The idea of the boldness of a theory can be easily expressed - the bolder a theory, the more states and behaviours of the domain it prohibits. Popper says '...theoretical science aims, precisely, at obtaining theories which are easily falsifiable in this sense. It aims at restricting the range of permitted events to a minimum' ([Popper80] pp41) and again 'Not for nothing do we call the laws of nature 'laws': the more they prohibit, the more they say' ([Popper80] pp113). In the case of the domain of electromagnetic radiation that was discussed in the previous paragraph, Maxwell's equations are more useful precisely because they prohibit a greater number of states than the single sentence theory stated above. If we have faith in Maxwell's theory, we can rely on never observing those behaviours prohibited by it. If we had access only to the less bold theory, we could not exploit the fact that nature appears to work in a much more predictable way than that theory suggests. The reason why the theory which prohibits more states and behaviours is a bolder theory is that it is easier to refute. There are many behaviours that the bolder theory prohibits that the less bold one permits. If one of these is reliably observed, then the bolder theory

[xxi] Although some question the 'honesty' of workers who claim to have used the mechanism [Feyer93], the author would contend that this shows that Popper's ideas should be understood as a framework for reasoning rather than a list of actions

is refuted. By asserting that a bold theory is a correct description of the domain, we are taking a greater risk that we shall be shown to be mistaken, but as long as we have not been thus discredited, our predictions will be more specific and thus more useful. The goal of science is thus not just to develop correct theories, but correct theories that are as bold as possible.

We are given two directives by the scientific method that together guide our actions. We must first create a theory that is as bold as possible, and then try as hard as we can to refute it. The scientific method has no contribution to make as to the relevance of the domain (as given by the interpretation of the instantiated specialisation state components of the theory) to the problem we are currently interested in. Maxwell's theory is bold and successful in the domain of electromagnetic radiation and has undoubtedly had a considerable impact on our lives. It will prove to be of little benefit to us in our current task of trying to understand the behaviour of clinical directorates. The difficulty of ensuring that the formally specified domain is pertinent to our problem is discussed below in Section 13.3.2.

The next two sub-sections explain how we interpreted the two scientific directives to the issue at hand - constructing a theory of the clinical domain.

The Experimental Process

To construct a theory of a clinical directorate is not in itself hard. Anyone who works in a hospital has his or her own ideas about what goes on and why. Formalising these ideas into a mathematical statement is not easy, but the nature of this task is not the subject of this thesis so we shall not discuss it here. Attempting to refute the theory is as we have seen an essential part of the process. Controlled experiments are not possible (certainly in this domain, and probably in most that are concerned with 'human activity systems'). We need some form of experimental process that does not need such controlled conditions.

The domain, as we have said, is not some form of objective reality[xxii], but rather a 'social construction' that is in some way shared by all interested parties in the domain (this idea is discussed in more detail later in Section 13.3.2). In this sense workers in a directorate are not 'impartial observers' in the 'scientific' sense but are aspects of and participants in the domain. The ideas, opinions and recollections that these participants have of the domain are more than proxy observations of some reality - they are, or at least can be, direct revelations of that reality itself (though we must not discount the possibility of a recollection or opinion being wrong). Accessing these ideas and recollections is possible, and is the basis of the experimental technique. The purpose of the experimental process is to provide counter-examples with which we can refute the theory in question: one of the directives that the scientific method provides us with is that we should pursue our objective of refutation with the utmost vigour. This means that experiments should test those aspects of the theory with which we have least confidence, or seem to be most surprising in their predictions. We must therefore seek out surprising or unlikely properties of the theory: these are expressed as theorems - rules that we claim to hold true in the domain.

Empirical science exploits this process and it is only those that have been tested most rigorously that are accepted as 'true' by the scientific community. Newton's theories of mechanics have been proved in this manner a number of times (and in this century found wanting). One of the surprising predictions the theory made was the existence of an eighth planet. For many centuries astronomers had known about seven planetary members of the solar system. It was felt that this catalogue was complete, and there were several explanations propounded as to why there could only be seven solar planets. The nineteenth

[xxii] It should be pointed out that in many ways the idea of scientific objectivity too is flawed: observers can never be totally detached from their observations, both in terms of their social constructions of the concepts and categorisations they are using, and in terms of their dynamic influence of those observations, a property most famously described by Heisenberg as his 'uncertainty principle'.

century saw the introduction of more powerful telescopes meant that planetary orbits could be studied more closely. In the course of these investigations a previously un-noticed perturbation in the orbit of the (assumed to be) outermost planet Uranus was discovered. Newton's theory predicted that the perturbation could be explained by a number of external influences, the most simple of which was the existence of an unexpected eighth planet with a certain orbital radius and mass (and hence size). When astronomers searched the part of the sky prescribed by the theory they found a shadowy object that we now know as Neptune.

Just as we can find surprising properties of a scientific theory, so we can of a theory of a human domain such as the one we have constructed. The experimental process used in this project involves deriving such theorems and endeavouring to elicit a counter-example from a participator in the domain through means of discussion and interview.

A crucial point should be made here. In order to elicit a counter-example from a worker in the domain it is not necessary to present or explain in detail the theory itself. It was decided that the mathematical structure of the theory as developed in this project should never be shown to a clinician - it is not important for him or her to understand the abstractions and behavioural structures in the theory - he or she is only required to give counter-examples of theorems to the analyst. It was considered that the best way to do this was to induce as broad a discussion as possible in the behavioural area which the theorems in question affected. In a typical interview the interviewee was not presented with an English statement of a theorem to refute, but was encouraged to describe his own understanding of a particular area. By endeavouring to draw counter-examples from the participant being interviewed, we avoid the necessity of having to teach him or her about the theory. Valuable contact time with workers in the domain is thus spent learning from those workers rather than teaching them how the analyst thinks. It is the author's contention that many analysis techniques suffer from this drawback where the major education effect is of the domain worker rather than the analyst.

Although the interviews described, being little more than a guided conversation with a clinician, seem about as far away from the formal scientific experimental process as it is possible to get, they nevertheless are based around the same philosophical concept. This is the desire to find counter-examples (or other refutative evidence) to the properties embodied in the theory: particularly properties which are in some doubt or seem unlikely.

An example of the elicitation of a 'counter-example' from a discussion was that which resulted in the abandonment of an early type structure to represent the behaviour of the day centre. The author presented some of the findings of the theory at a departmental research meeting. The areas that were chosen for presentation and discussion were those that were open to some questioning, although of sufficient credibility to be worthy of consideration by the departmental staff. Thus the type structure of the clinic was discussed, and the life-cycle of activities was not. There are valid theorems that could be clearly discussed with clinicians concerning the history of an activity. For example an activity that has been completed may not re-commence. This property did not seem particularly controversial or a useful subject for debate and so was not dwelt on. The hierarchy of types on the other hand implied a behaviour in the clinic which the author believed to be true (it had already been 'tested' in earlier interviews) but nevertheless was quite surprising. This was that the paramedical staff had always completed a sequence of consultations before the next doctor consultation. It was asserted by a clinician present that there were cases where the paramedical staff dispensed care totally separately from the doctor, and thus the two processes - the six-monthly review of the patient by the doctor and the periodic encounters (often patient

instigated) with the specialist nurse or dietitian - can run in parallel. The details of this 'experiment' were discussed in Section 9.5.3.

While the setting up of interviews for the purpose of exploring a particular area of the theory is similar in many ways to the experimental process, there are other benefits to be gleaned from discussing the domain with its participants. During discussion an entirely unrelated topic or fact might be mentioned that the analyst had not intended to investigate further, and yet refutes one of the theorems of the theory. An example of this might be the discovery that some activities seemed to have more than one patient in attendance at a time. That an activity referred to only one patient was not open to question. However, while discussing the activities of a specialist nurse with a doctor, it transpired that the 'patient education sessions' which they ran were generally attended by half a dozen people[xxiii]. The way that this refutation was dealt with is explained more fully in Section 10.2.2.

The interview, being a form of experiment, was the most important refutative mechanism used in the project. There were others however, the most important of which are 'refutation by observation' and 'refutation by inspection'. Refutation by observation takes place when a counter-example to a theorem is observed directly by the analyst. For example, in a very early version of the theory (before the first attempt at formalisation) all clinical activity had been divided into tests, anamnestic consultations and interventions. Sitting in a clinic session with one of the clinicians, it became apparent that any such distinction was meaningless as each consultation consisted of all three - testing the eyes and blood pressure one minute, taking a medical history the next, and advising the patient on their lifestyle (education is one of the most important interventions in the care of diabetes) the next.

Refutation by inspection is the last refutative mechanism used in the project. While the 'experimental' interview is the most important conceptually, refutation by inspection is the most useful. This involves the use of knowledge accumulated by the analyst to see if he or she can find a counter-example to the theorems or laws implied by the theory simply by looking at it. An example of this was the refutation described in Section 9.3.3. It was realised that the insistence of resource sharing for any ordered activities meant that 'Blood Tests' where phlebotomy preceded analysis were disallowed. The analyst did not need to ask a clinician about the validity of this invariant - it is clearly invalid. Through inspecting the theory a property was 'discovered' that was deemed incorrect based on the analyst's own understanding of the domain. Care must be taken here however - it might be obvious to the analyst that a theorem is incorrect but the analyst could well be wrong in his or her assumptions. The construction of the domain by the analyst is inevitably going to be very different from that of the clinician, and it is for this reason that the 'experimental' interview is the most important mechanism.

The Boldness of the Theory

We have seen that there are a number of mechanisms that we can use to refute the theory that has been developed. This will help us move towards a theory that is 'correct', or at least more correct than one that has been refuted. This alone is not enough - as we have seen, a theory can be correct and still useless. We need to enhance any theory to forbid as many behaviours as possible (within the limit of refutation) - it needs to be made as bold as possible. There are two ways in which we can do this: by the introduction of new constraints on existing state components; and by the introduction of new state components that interfere with existing ones. We shall see how these two forms of theory enrichment were used to develop the domain theory that has been presented.

[xxiii] It would appear that this is a fairly common mistake to make. The operational requirement of the Outpatient Appointment System for St Thomas' Hospital specified that an appointment slot should never be associated with more than one patient.

An example of a case where existing state components were tied together by invariants so as to constrain the number of behaviours possible was the specification of the interaction between the graphs over activities: *Before* and *After*. When the theory was first constructed no constraint between these two relations was defined. As the analysis progressed, it became clearer what it meant for an activity to be *Before*, or *During* another. To represent this emerging understanding a greater semantic richness was introduced into the theory. This semantic richness was precisely represented as a reduction of possible behaviours. This reduction of behaviours was in turn represented through the specification of certain states as being 'illegal': in short, through the use of invariant properties that could not be violated. Firstly it was noticed that the precedence of activities did not really apply to activities that were during one another - thus all states where an activity is in the relation *Before* with one of its ancestors through *During* are disallowed. A subsequent realisation was that the conventional understanding of *During* incorporated notions of encapsulation - if an activity was *Before* another, it was also *Before* all those activities that were *During* it. Thus all states where an activity is *Before* any others that are not 'siblings' were declared to be illegal. All states that are excluded by the former prohibition are also excluded by the latter, and so one invariant can express both properties - this invariant is I11. Both of these prohibitions constrain behaviours without the introduction of new state components. The theory without the invariants was just as 'correct' in that there are no behaviours allowed in the constrained theory that were disallowed in the unconstrained one: if the constrained theory is unrefuted, then so too will be the looser one. However, since the more highly constrained theory has a greater degree of semantic content by virtue of the introduction of the prohibitive rules it can be said to be a bolder, and so better, theory.

Another case where existing state components were constrained so as to reduce behaviours was discussed in Section 9.4. Here the existing state components are *Activities*, its subsets, the various graphs over them, and *Types* and its structures. At one stage of the theory's development, the only constraint involving these components ensured that any activity structure could be projected onto firstly *Can_include* and then *TypeGuide* (given by invariant I32). This property was not refuted, but still allowed what seemed to be unrealistic behaviours. According to the theory, not only could an activity of type Dietitian Care include one proceeding activity of type Dietitian Consultation, it could equally contain many activities of that type, all of which were proceeding. It was asserted that this is not observed, and two invariants covering the general case were introduced. These were invariants I26 and I27 which state that no two siblings (through the *Includes* relation) can both be proceeding and of the same type, or both be requested and of the same type. Again, this constraint does not add to the scope of the theory but through the prohibition of certain states (and hence behaviours) increases its semantic content.

Not only should we strive to constrain the behaviours allowable by the theory within the limit of refutation, but when a particular theorem is refuted, we should ensure that a new theory, while allowing the existence of the observed behaviour, is still as restrictive as possible. An example of this is discussed in Section 10.2.4. The theory originally stated that a patient could be physically present at only one activity at a time. The example of the chiropodist consultation, at which a patient must be present, including a Dr Pop-in which the patient must also attend, refutes this. Instead of abandoning the invariant altogether, allowing patients to be physically present at any activities (that they are the subject of), the new theory allowed patients to be present at more than one activity only if they are all direct ancestors or descendants of each other. In this way the counter example that refuted the theory is accommodated but the increase in the number of behaviours allowed is minimised.

The above three examples illustrate how invariants were introduced to the theory so as to reduce the number of possible states that models could exist in. We can also make the theory 'bolder' by introducing new state components that interfere with ones that already exist. An example with this is the introduction

of first the relation *Can_include* and then the structure *TypeGuide*. By the time the theory has been refined to ATClass1, all activities are defined as being created with a type that does not subsequently change. At this stage an activity's type is not constrained in any way other than to insist that it is a member of the set *Types*. The next refinement introduces a new state component and associated invariants that constrain the number of behaviours. The finished theory uses the structure (a triple) called *TypeGuide*, but the ideas behind its introduction are equally well illustrated through discussion of an earlier theory that used the graph *Can_include* and we will thus use this relation for simplicity.

Can_include is an acyclic graph over types that does not change after being set up (in this version of the theory). As each activity has a type, we can project a structure of activities into type space. The invariant that is introduced with this new state component insists that the projection of any permissible activity structure into type space is a subset of *Can_include*. By introducing this new state component and its associated invariant we massively reduce the number of possible states of a model of the theory. We are forced to reduce the number of allowable states no matter what (legitimate) values we give to the relation *Can_include*: as *Can_include* is an acyclic graph, there are more pairs of types that are excluded than are allowed (ie $2 * \#\ Can\_include \leq \#\ Types \times Types$ for any allowable sets *Types* and *Can_include*). This is a case where a new state component was needed to represent the constraint we wanted to describe, but the number of possible states of models of the theory after the introduction of the new state component is less than it was before.

It is generally not the case that the introduction of a new state component that is not constructed from existing carrier sets constrains existing behaviours. *Can_include* was constructed from an existing carrier set, the set Ta. The introduction of types also introduced that carrier set, and so we would not expect the appearance of *Types* in the theory to constrain behaviours. We might take the introduction of the set Patients as an example of this case. There are all sorts of invariants governing the interaction of activities and patients, such as the association of every activity with exactly one patient, and the insistence that every activity in a given structure is associated with the same patient: these do not represent reductions in the volume of state space over the theory before Patients was introduced. In fact the introduction of a new state component massively increases the number of possible behaviours that a model of the theory can exhibit: for every valid state before the new state component, there will be at most the cardinality of the new set valid states after its introduction. However, if we put any constraints on the interference of the new set with the existing state components, we have still managed to embolden the theory. The reason for this is that we should not measure absolute numbers of states prohibited, but rather the ratio of the 'volume' of state space forbidden to the volume of state space permitted. If we introduce a new totally disjoint state component that behaves independently of the existing system, we have not changed the ratio of the number of forbidden states to the number of permitted states meaning that the theory is no bolder than it was. If we define an invariant that acts to mutually constrain the states of the existing system with the new set, then we have moved some of the previously permitted states into the forbidden region of state space, thus increasing the ratio of forbidden to allowable states. By doing this, we make the theory easier to refute: for every state that was allowable before, there are a number of new states that are forbidden (where the previous state components keep their allowed values, but the new set takes values that are prevented by the mutual constraints imposed by the invariants). Because the theory is easier to refute, it is bolder. The introduction of new state components that interact and interfere with the existing system is something that should be welcomed, and is in keeping with the general theme of theory emboldening discussed in this section.

The construction of the theory was driven by the desire to create a description of the domain that was not just unrefuted (after all reasonable attempts to do so), but that was bold and thus a 'realistic' and useful

reflection of the perceived reality. The previous part of the thesis that presents the results is constructed in such a way as to represent the increasing enriching and thus emboldening of the theory. For further examples, the reader is particularly referred to Section 10.3.5 that describes the introduction of the set *HCP*, and Section 10.5.5 that describes the representation of organisational boundaries.

Conclusion

In the last section, we have discussed the conceptual underpinning to the method used to derive the domain theory that is presented in Appendix 2. We have shown how the scientific method as originally defined by Sir Karl Popper acted as the basis of the approach. The method of refutation of existing theories has been modified so that the controlled experiment has been replaced by the 'experimental interview', and examples of this were discussed. The development of the theory was guided by the second 'scientific imperative' - the desire to create a theory that is as bold as possible in that it prohibits the maximum number of behaviours or states of models of the theory.

In the next section we will consider what shortcomings and problems with the method were encountered over the course of the project.

13.3.2 Criticism

In this section we will consider some counter arguments that suggest the approach described above to the derivation of a domain theory is at best simplistic, and at worst fatally flawed. The arguments presented can be summarised as follows. The conduct of a refutative 'experiment' can be extremely difficult if not impossible for some theorems, and might anyway give misleading results. The development of the theory, although encouraged, is in many ways arbitrary as far as the described method is concerned. The 'constructed' and personal nature of reality, although recognised in the method, causes deeper problems than have been addressed. Finally, mathematics might be a wholly inappropriate medium with which to express such a socially complex domain as a clinical department.

The Difficulty of Refutation

There are a number of reasons why refutation can be extremely difficult. There are many reasons why we might not be able to refute an incorrect theorem - this sub-section lists a few that were discovered over the course of the project.

Firstly some behaviours can take a very long time to refute. This might be because counter-examples are observed very infrequently in which case we should not necessarily worry[xxiv] - models of the theory, and hence well-derived information systems, will normally behave as expected if this is the case. More awkwardly, it might take a long time to observe a refutative example (or elicit one from a domain participant) because the behaviour it applies to is very slow. A case in point is the evolution of the clinic itself in terms of its operating procedures and organisational rules. This was discussed at some length in Section 8.3.5, but will be recapitulated here to support the current argument. Although the theory does not try to explore these processes of organisational evolution, it does try to represent valid states of organisational structure at (fixed) moments in time. This it does through the use of the various invariants governing the interaction of *Types* with itself and *Activities*. Although it is possible to describe the state in a single clinic (as represented by, for example, the specialisation of *Types* and its structures to the DEDC), it is much more difficult to identify general rules that apply either to a number of different

[xxiv] Note that in certain cases it is the infrequent events that are the most significant. A diagnostic system that failed to spot a particular condition merely because it was rare would be of dubious benefit. It is considered that these 'rare' cases are less significant in the realm of organisational & administrative behaviour.

organisations (remember that before specialisation, the theory is supposed to be general for a number of departments), or to the same organisation as it develops over time. In order to refute a general theory about the organisational structures of clinical departments, it would be necessary to construct specialisations for a number of these different medical areas. Although other clinical domains were investigated (particularly the Diabetes department at the Medway hospital and the Dermatology Directorate at St Thomas' Hospital), only the DEDC at St Thomas' was explored in any great detail. Even if we did manage to construct a specialisation of different directorates or clinics, it would not be clear that a refutation was the result of incorrect rules governing the structure of all medical domains, or of a poor specialisation. In other words we would not know whether the values of the set *Types* and its structures, or the invariants governing them were incorrect.

Another reason why it is difficult to refute a theory is the inevitable blindness of the analyst when it comes to understanding the meaning of an interviewee during one of the 'experimental' interviews discussed above. This is similar to the idea of paradigm traps discussed below only more immediate. We have seen that we do not want to explicitly represent the user's concepts of the world she inhabits, and that we want to elicit counter-examples to the theory we are developing so as to be able to construct models that exhibit familiar behaviours. It is thus up to the analyst to understand which concepts in the theory describe the entities being discussed by the participant. This can, to a certain extent, be avoided through the use of the implicative mechanism of the mathematics. When confronted with an example of a behaviour, the analyst can question the interviewee about a property implied by that behaviour taken with the rules in the theory. It is still easy to make 'mistakes' as a result of assumed interpretations of the behavioural descriptions elicited from the domain participants. A good example of this is the 'DNA' activity which the theory does not currently represent. The reason for this is the difficulty of finding out what concept is being discussed when we talk about such a thing. A 'DNA' is a 'Did Not Arrive' - a visit or consultation an expected patient did not show up. These are genuine entities in the DEDC as construed by its staff and statistics are collected which detail how many 'DNAs' there were, and to which District Health Authority they could be assigned. It is not clear however, which abstract concept such a thing belongs to, or whether it needs a concept of its own to describe its properties. For example, it might be a type of activity in which case what was a request of one type would become, on the non attendance of a patient, a proceeding activity of another type. Alternatively it could be an attribute that could be attached to any activity that patients are able to attend. The reason why the latter approach was not used is because when questioned in an interview, one of the clinicians was emphatic that the DNA was a sort of medical encounter, only with the patient absent, and medical decisions might be taken in it. From this explanation it would seem that a DNA was indeed a type of activity with all the attendant problems - common sense would tell us otherwise however. It was not apparent what the appropriate course to take was - questioning the clinician further would quickly have resulted in confusion for all concerned. As a result, the problem was put on one side to return to at a later date (it is listed as one of the outstanding tasks in the conclusion to Chapter 10). This might seem like a trivial example, but is one which the author has noticed: in general it is very difficult to identify areas where incorrect, or dubious, interpretations of the discussions with stakeholders have been made. It is clear however that such problems of interpretation inevitably exist.

A reason for difficulty in the accurate conduct of the experimental process as described will be obvious to the reader from the rather dry way in which the scientific method has been presented. Science is used to manipulating and forcing the natural world so that it answers the questions that are of interest. Francis Bacon described this tendency in a delightful and famous quotation:

'... if any expert Minister of Nature shall encounter Matter by mainforce, vexing and urging her with intent and purpose to reduce her to nothing; she, contrariwise ... being thus caught in the straits of necessity, doth change and turn herself into diverse strange forms of things. ... the reason of which constraint or binding will be more facile and expedite, if matter be laid hold on by Manacles, that is by extremities.'

The rather fiendish language used should not bother us in the normal course of science (at least the physical sciences - we, though maybe not Bacon's contemporaries, would blanch at the application of this attitude to, say, animal experiments) - it is if anything an amusing thought that a controlled scientific experiment can be seen as the torture of Mother Nature. It would no longer be amusing if we understood that the subject matter of our experiments was living people rather than inanimate 'brute matter'. The consequence is that the experiments that we claim to be conducting through the structured interviews described are almost totally uncontrolled. There are all sorts of ways in which the 'results' we obtain might be erroneous. An obvious one is through the deliberate misleading of the analyst by the interviewee. This is particularly likely if the subject matter is politically highly charged. This is one of the reasons why the analysis confined itself to the less contentious area of the operational behaviour of the directorate. Even so we should remember that no human activity is ever totally apolitical. More prosaically, the interviewee might be disinterested in the line of reasoning being pursued in the discussion, or unable to see the point, or just annoyed about being badgered repeatedly about the same issue. All of these will result in unsatisfactory elicitation of refutative counter-examples. Having made these points, the author would like to stress that in the course of his analysis, all the domain workers who were interviewed were extremely helpful and forthcoming with information, and if they were bored by a particular line of argument, never showed it. This might not always be the case however, and the geniality of the eventual users of a computer system is certainly something that cannot be relied on.

Lastly, we will revisit a problem that was discussed previously in Section 10.3.4 - that of the paradigm trap. The term is taken from work by the philosopher Thomas Kuhn [Kuhn70]. Kuhn said that Popper's presentation of scientific progress was overly optimistic and orderly. Rather the course of science could be viewed as a sequence of revolutionary paradigm shifts, between which normal science (as described by Popper and his followers) proceeded, but without which human knowledge would not progress significantly. A paradigm is a way of thinking, a 'conceptual framework within which scientific theories are established' [Chamb89] - great leaps forward in science are enabled by the discovery of a new and more powerful paradigm which we can use to think about a particular part of science. The classic example that is often given is that of the replacement of Ptolemy's cosmology by that of Copernicus. While Ptolemy's model of the cosmos, with the Earth in the centre and the planets and stars on epicycles circling it, had been adequate for hundreds of years, it was holding back the development of the subject. In order to make its predictions more accurate, more and more complexity had to be introduced to the basic model until it became essentially unworkable. This is not to say that the theory could not or did not develop. It was the basis of astrological and astronomical calculations until late in the fifteenth century - if an error was identified the theory was refuted and a new one proposed which would generally be identical to the previous save for the addition of a new epicycle. It was not until the Copernicus published his *De Revolutionibus Orbium Coelestium* in 1543 that this paradigm was replaced with the familiar one which has the sun as the centre of the solar system, and the earth as one of several planets orbiting it. The advance that this new paradigm represented cannot be understood in terms of the refutative process, or even of the need to 'embolden' the theories that have been developed using the old paradigm, as both of these were in evidence during the pre-eminence of the Ptolomaic system. However, the conceptual clarity embodied by the Copernican theory led to a new golden age of astronomy, enabling the discoveries of Brahe, Kepler, and eventually Newton. The history of science is punctuated with these paradigm shifts

which act to release intellectual bottlenecks, and galvanise progress in its composite disciplines. Further examples from physics are the creation of quantum physics by Böhr, Heisenberg and Schrödinger, and the two theories of relativity, special and general, proposed by Einstein. An example from molecular biology is the discovery of the helical structure of DNA by Crick and Watson and the associated 'decoding' of the genetic message. What this tells us is that there is another mechanism at work in the construction of theories of the world over and above the scientific method described which is more akin to inspiration than conventional scientific endeavour. The method espoused by Popper cannot help us find new and more powerful paradigms, nor can it tell us when an existing one has been exhausted. We can only rely on our own feeling for elegance, simplicity and clarity for that.

An example of a paradigm trap that the author 'fell into' is that concerning the generation of new activities. The paradigm that was initially used was one which viewed the activity as being the creator of new activities. Any activity that was created must have another activity as its creator. Not only must an activity have a creator, but that creating activity must be guided by rules over types: certain types of activities would only be able to create certain other types, depending on the circumstances. Rules were proposed and defined as invariants which structured this behaviour. In order to reflect the subtleties of referring authority within the directorate, these rules became increasingly complex. A simplified account of the evolution of the behavioural constraints on the creation of activities was presented in Section 10.3. In that section we saw that a structure called *InLoco* had to be introduced which allowed one activity to act as if it were another. This needed to be changed from a relation to a triple, and the attendant invariants defining its interaction became increasingly confusing and unwieldy. Even with this greater complexity, refutative counter-examples could still be found. The prospect of the need for structures and rules of ever increasing elaboration and decreasing clarity presented itself. One solution to this would have been to have admitted defeat and said that the constraints on the creation of new activities was outside the scope of the description embodied by the theory. This was considered unsatisfactory as the structure of activity creation was perceived to be at the heart of the intended domain of discourse - the operational behaviour of collaborative clinical groups. In the event this was not necessary as it was decided that a new paradigm could be used to guide the description of the relevant processes. This involved the rejection of the idea that activities begot other activities, and replaced it with one where health care professionals acted as the creators. With this new paradigm, although some of the structures are of a similar complexity, the author feels that a greater degree of conceptual clarity has been achieved, facilitating future 'progress' both in terms of accuracy of representation, and in terms of concision and lucidity.

Although new paradigms for understanding parts of the domain can assist the development of the theory greatly, the 'scientific' method espoused does not help us to discover what these might be and when we might need them. Indeed, it might be argued that the labour associated with representing the consequences of the use of a new paradigm in formal notation discourages us from looking for one. However, we cannot delude ourselves when a particular aspect of the theory needs a different underlying conceptual structure. Some examples of areas in the current theory that are in need of new paradigms are given in the next Chapter.

An Arbitrary Description

There are two senses in which the description represented by the current domain theory is an arbitrary one. Firstly we can find nothing in our method which tells us what an appropriate 'scope' of the theory might be, and secondly we are presented with no advice on how to represent the phenomena we observe and elicit from the domain participants.

To take the former point first, we do not and cannot know what the scope of the theory should be. In other words, we have no way of knowing whether an observed property of the domain should be included in our description or not. This is not an omission, but a fundamental problem of this approach, and indeed of systems analysis in general. The essential problem is that we are endeavouring to describe a part of the world and not an information system. A central tenet of the method used over the course of the project is that we cannot understand the nature of an information system that will in some way support the organisation before we understand the nature of that organisation. Similarly the particular part of the organisation that we want to represent will depend on the function of the information system. Clearly this is a 'catch 22' situation. However, we cannot avoid focusing on a particular area of the domain for to describe 'everything' is infeasible. We must make some broad assumptions therefore about the sort of information systems we are eventually hoping to design, and guide the scope of the description on the basis of those assumptions. We saw how some assumptions of scope preceded the formal phase of the analysis in Section 7.3: we cannot be sure that those assumptions are valid, and even so they only provide a measure of guidance and to a great extent we are still 'groping in the dark', relying on intuition and experience. Having accepted the assumptions of Chapter 7 (namely that we should describe a clinical area as generically as possible, concentrate on operational behaviour, and avoid medical details), we might describe an area of the domain in insufficient detail, or equally in too much detail. An example of each of these cases is presented below.

Firstly we might not examine a particular part of the domain in sufficient detail. It is difficult at present to say categorically where the domain theory is inadequate in this respect. Some areas where more work could usefully be done are booking (so as to see whether the hospital's OPAS booking system is overly prohibitive as discussed in Section 12.4.6) and clinic lists. The details of these cases are complex - for the sake of example we will take the straightforward case of medical exclusion of certain types of activity which clearly illustrates the points raised. The specialisation of the theory to the DEDC makes reference to a type of activity called Diabetic Pregnancy Care, and to another called MARS Care. The MARS clinic, which is periodically held in the centre, is for men only being concerned with issues related to male impotence. Similarly the Diabetes Pregnancy Care activity type only concerns women. The theory as it stands does not feature gender at all[xxv], and consequently allows for unrealistic behaviours. For example, a patient is allowed to attend MARS sessions while being given pregnancy related care. This is clearly an impossible behaviour - although the theory does not forbid behaviours that are observed which would mean that we would have to re-examine our understanding of the domain, it does allow behaviours that will manifestly never be observed. In this sense the theory could easily be made bolder and hence better. Clearly one has to stop analysing the domain somewhere if an information system is going to be designed in a finite time, but where we decide to effect that stop is an entirely arbitrary decision. In this case a reasoned argument could be made justifying the decision not to get involved in this consideration of which activities prevent which others - for one we would very quickly get caught up in the morass of 'clinical knowledge' for which many have tried and failed to provide a valid generic theory. However, there is nothing in the method described which tells us when we should explore further, and the decision to stop analysing at a certain point is entirely informal.

An example of an area where perhaps too much effort was expended was in the issue of internal referrals and activity creation discussed above in Section 10.3. Although a reasonable (albeit possibly overly complex) theory has been defined and presented to describe behaviours in this part of the domain, the arrival at the final version of this took a great deal of time. If it had been decided that this was outside the

[xxv] In fact, gender has been specifically excluded from the theory on clinical grounds related to the nature of endocrinology (see the un-numbered section titled 'Conclusion to Chapters 8, 9, and 10').

scope of the theory, other areas could have been tackled which would have provided a greater degree of insight. Of course it is impossible to know in advance how much effort will have to be invested in order to get a good description of a particular facet of the domain's behaviour, but in hindsight we can be wise and say that perhaps other areas would have been more fruitful if attacked with the same amount of vigour.

In short, in both these cases we are given no help in knowing when to stop our analysis - the questions 'have we already gone too far' and 'have we not gone far enough' are not answered at all by the method used, and inevitably many will deem the decisions that were taken to be poorly considered.

There is another way in which the description is arbitrary apart from questions of theory scope - this is in the structure we choose to represent the phenomena we observe. The problems of interpretation - knowing what perceived or described entities to assign to which concepts has already been touched on and will be discussed again later on in this section. There are more pernicious problems here however.

Firstly we have made assumptions as to the nature of operations. An operation can be regarded as a node in a tree of possible behaviours of a model of the theory. The number of possible states resulting from an operation depends on the pre-state and the particular operation invoked. It might be that a single operation that allows for a profusion of post states could more reasonably be represented as two distinct but more restricted operations. An example might be the ***.SuddenStart*** operation which creates a new activity that is in the set *Proceed*. The decision to include this single operation instead of a normal ***.Create*** followed by a ***.Start*** is fairly judgmental, being justified only by the assertion that for certain types of activity the state where that activity is a request is never observed. This argument could easily be reversed however, and we might say that all activities are initially requests, but some for an infinitesimally small time. Similarly the categorisation of operations is down to personal preference. We could have chosen to represent the ***.Start*** and ***.End*** operations as one, called perhaps ***.ChangeState***, which moved an activity to the set *Proceed* or *Complete* depending on its current status. These are fairly trivial issues, but we should be aware that they exist, and that we have made value based decisions with regard to their outcome.

A more subtle problem concerns the decision to represent a particular phenomenon as a state component rather than an operation, or *vice versa*. We have chosen to describe the part of the world we are concerned with in terms of values of state components that can be interpreted into the world. The operations we define describe how that state changes. There is a clear separation here between 'object' and 'process'. If we look more closely at the objects, we might be able to imagine representing some of them as processes, and similarly if we investigate a particular process described by a single or sequence of operations. To take the first case, many would find the terms 'operation', 'event' and 'activity' fairly synonymous. Why have we chosen to represent one of these concepts, activity, as a state component? The activity is a process - it is something that happens - and to many people it might be a completely counter-intuitive notion to represent such a thing as an object. The notational structure chosen does not require processes to be instantaneous - only that we describe the state of the system before one may start, and following its successful conclusion. Indeed we could very well choose to think of the activity as an operation (and indeed some will certainly be 'operations' in the surgical sense). If this decision had been taken from the start the structure and feel of the theory would be very different. The choice made in this respect seems to have worked reasonably well, but we must again be aware of its arbitrariness, and alert to possible alternatives.

It is interesting to note that these issues are reflected in philosophical ideas and movements. For example, the Greek thinker Heraclitus eschewed the notion of the world having a state which was subjected to

change. Indeed he is quoted by Plato as claiming that 'all is flux' (described for example in [Russell89]). The notational framework chosen does not readily support this way of looking at the world (at least in the way it has been used in this project), but others (for example CCS and CSP) would do so more readily. The fact that questions concerning the representation of a thing as a process or an object have been discussed since the ancient Greeks indicates the persistence of the issue. The debate is still relevant to the area of medicine. In an early version of one of the 'feeder' projects for the Common Basic Specification of the NHS (DiabPTech), the 'object of care' was represented as a process to reflect the dynamic nature of the living organism:

'...the object of care ... is represented as an activity. This reflects the reality that an object of care continually changes for example the process of ageing. This difficult notion is necessary if we consider populations which are continually in a state of flux, or consider biochemical or physiological components of the object of care which are never static.' [Harrison89].

In none of these above areas - the question of theory scope, the identification of operations, and the division between state and behaviour - does the method used help us to make the 'right' decision. If we decide to represent an area, it can tell us whether that representation is a good one - it cannot help us make these fundamental decisions which are in many ways arbitrary and a question of personal taste. We must be aware that these decisions have been taken, and be prepared to review them if necessary.

There are more serious issues that we must confront if we are to be honest about the utility of the method. These concern the problems with the very idea of representation - can we ever hope to derive one representation, and can such a representation ever hope to be a mathematical one.

The Social Construction of Reality

Classical philosophy has always maintained the duality of nature. This duality consists of an absolute physical reality on one side and the human mind and spirit on the other. Over the past century, various ideas have come to challenge this doctrine which is still very firmly embedded in the western mind. A number of schools of thought have established themselves which have very different perspectives on the universe, some questioning whether we can even say that such a thing exists, or more extremely whether existence is a meaningful concept. Although there have been 'meta-analyses' conducted (for example [Bick92]) which attempt to present an overview of large numbers of analysis methods and the philosophical stances that inspire them, this is certainly not the task of this thesis. However, we should at least recognise that the 'classical' understanding of reality is flawed, and any analysis method, especially in the area of computing science, should not adhere too slavishly to it. One interesting and widely supported philosophy has it that the reality that we see around us is essentially constructed by our own thought processes. This view has been accepted by many in the academic computing community [Floyd91], and has formed the basis for a number of analysis techniques (two typical widely used examples are SSM [Check90] and ETHICS [Mumford86]): in this section we will see what problems such an approach presents to the user of the 'scientific' method described here, and how these problems manifested themselves over the course of the project.

One way of understanding the constructed nature of reality is as follows. In a way it is nonsense to maintain that reality is constructed: we cannot solipsistically deny the existence of any world outside ourselves - what happens happens, whether we like it or not. The social construction comes with the construction of an intellectual framework with which to view and think about the world. The grouping of a number of molecules into an entity we call a person, the attachment of the label 'molecule' to an object, the segregation of what we see into object and process, the categorisation of a large number of mental

impulses as 'sight': none of these is a property of 'reality out there', but of the way we understand, interpret and think about the universe in which we find ourselves. Having said this, the basic understanding of the world, the naming of basic categories is fairly constant, at least in a single society. Different societies may well have very different ways of categorising their 'universe' and consequently the brand of reality that members of that society inhabit might well be different from that of our own. Any group of people that we call a society will be composed of a number of what we might call 'sub-societies' which in their turn will comprise 'sub-sub-societies' and so on. At any level of division the (arbitrarily selected) groups will have their own slightly or radically different realities. If we look at the hospital, there are a number of possible sub-societies - dividing the staff into different directorates is one way of forming these separate groups as is dividing the staff by profession, discipline, age, sex and so on. One of the sub-societies consists of the workers in the DEDC. We can keep sub-dividing the societies of people to whom we ascribe a reality until we get to the 'atom' - the single person who will have a different reality from each of his neighbours (at least according to this philosophy - the ideas here are clearly not susceptible to proof). Although each person that belongs to a society will have a different reality, some of their concepts will be shared as a result of the interactions they are forced to make.

According to the philosophical stance briefly outlined above, although to a great extent there will be many concepts shared between workers in the DEDC, each will have a (possibly subtly) different reality. The theory cannot represent more than one 'system' at a time which means that we will be forced to choose a particular reality to represent. If we are correct in our assumption that the DEDC is a well defined 'sub-society', we can try to capture some of the essence of the shared reality in the theory. In this sense we can expect at least some stability of representation. However, we will inevitably come across some disagreements between domain participants that are the result of their different realities rather than mistakes and omissions having been made in the describing of those aspects of reality that are shared. It is very difficult to identify where this might have occurred, but it inevitably has. One area where different realities encountered will be made apparent is through the different extensions of concepts with the same name. The classic example in the health service is the contents of the set named 'bed'. To some people in the health service, a bed is a physical construction on which a patient lies. To others it might be an area of the ward - 'beds' stacked in the store room would not be beds at all in this case. For some people a hospital trolley would be thought of as a bed - generally this would not be the impression of the patient lying on it.

This lack of co-extensivity of named concepts is one area where differently constructed realities manifest themselves. There are other more subtle areas however. One encountered during the course of the project concerns the hierarchical nature of medical care. Two different views were presented to the analyst that could not both be accommodated in the same theory. The first was that the treatment of the diabetic patient revolved around the doctor visit. The doctor, having overall responsibility for a patient, managed him or her by means of a series of visits. As a result of these visits, the doctor might decide that a course of treatment, therapy or education could be delegated to a paramedic, but this would be at the instigation of the doctor concerned. The theory at this stage reflected this fairly rigid hierarchy with all paramedic and specialist nurse activity being a component of the extended doctor visit. A different view was presented to the author a short while later. On this occasion the (different) interviewee maintained that the different health care professionals all acted fairly autonomously: a specialist nurse might well see a patient completely independently of doctor. These two views need not be different tales, only one of which can be correct, but might reflect two different understandings of the universe in which the domain participants exist. If this difference is indeed the result of two contradictory realities, it would be impossible to decide objectively which of these two views should be represented in the theory. It is not possible to represent both however, and so in this case a subjective decision was made. The details of the different theories were discussed in more detail in Section 9.5.3.

Another, perhaps knottier, problem that faces us, in our attempt to understand the 'shared constructed reality' that we believe can be deduced from discussions with stakeholders, is that it is in general impossible to establish the interpretation that a domain participant has of a particular concept. What this means is that when we discover what appears to be a refutative example showing a state or class of states that are forbidden by the theory, we cannot know whether this is because the structure of the theory is wrong, or because the intended interpretation of terms in the theory is not that which an interviewee has made. This problem is most acute when it concerns different extensions of the same concept as was discussed briefly earlier. The fact that a concept will be interpreted slightly differently by different people should come as no surprise - the difficulty is identifying where this has occurred. The concepts that are discussed by the different domain workers may be different, but if they have the same name, the lack of co-extensivity is not apparent.

For example, one interviewee might present examples of a property possessed by elements of the class 'Patient' from which we derive a number of theorems. Another interviewee might present examples of behaviour and state of the same concept that contradict the theorems derived earlier. This might be because there is a class of people that the first interviewee considered to be patients that the second did not. If one interviewee was a surgeon, a property she might ascribe to patients is their intermittent nature - a person can be a patient and undergo surgery, convalesce and leave hospital whereupon they are no longer a patient, or at least not until they are re-admitted. If the second interviewee was a diabetologist, she might deny that a person ever stops being a patient, or at least that everyone registered with the diabetes clinic remains a patient until they leave the area or die. These two properties seem to be flatly contradictory, and a degree of understanding of the medical domain is required, even in this coarse example, to see that although the same word is used - patient - the surgeon and the diabetologist are in fact talking about different concepts. This problem is a variant on that described earlier where different people have different 'realities': the fact that the two aspects of reality that differ have the same name makes it very difficult to pick up.

It was said earlier that examples of behaviour that refuted the theorems of the theory rather than agreement with general behavioural rules were elicited. As we are not concerned with agreement with classes of behaviour as defined by the theorems (derived from reasoning over invariant predicates and behaviour traces), it might appear that we should not have to worry about the interpretations of concepts by domain workers. However, closer investigation of the counter-examples supplied in interviews invariably reveals that single events are rarely if ever described - instead classes of behaviour are presented. We can see that this is the case by considering the outcomes of the various interviews. The following example of a discussion about 'shared care' illustrates the point.

'While some GPs may expect a full clinical service if the patient requires it, others do not expect this, and at least one (Dr X) expects that the patient is sent straight back to him as he has a sophisticated diabetology service run from his surgery'.

The interviewee, although being specific and not claiming to make any general statements about the nature of medical care, is nonetheless making reference to classes of behaviours and objects. The phrases 'some GPs', 'the patient', and 'a...clinical service' all refer to classes of object. Similarly the expectation of 'some GPs' is of behaviours that are similar and can be observed many times - ie the provision of a 'full clinical service', and Dr X (an individual, not a 'class of objects') expects, when certain criteria are satisfied, another form of behaviour (that the patient will be sent straight back) for all of his patients every time they attend the clinic.

This might seem overly pedantic, but it is important that we recognise that abstract concepts are always used in discussion (and not just with doctors, but with all people), and the problems of concept extension (and intension) will inevitably dog the analysis. Although the interviewee will use generalities rather than instances in conversation, we should encourage her to use concepts that are less abstract than those in the theory rather than enter into a direct discussion of the worth of any derived general theorems. An example where one class of behaviours was elicited from a domain worker and used to refute a general theorem concerns the number of activities associated with a single 'slot'. Here, an early theory maintained that a slot could be associated with only one activity at a time, and hence only one patient. In discussing this area of the theory with a clinician, it became apparent that for a number of activities, for example the patient education session, several patients were associated with the same slot. Here both the theorem and refutative observation refer to classes of behaviours, but the refutation applies to a much smaller and more specific class than the theorem being refuted. Because the class of behaviours being described is smaller, we are justified in attributing a greater degree of credibility to it.

There is a further problem that we have not covered yet but is particularly relevant to the method we have used. This is the use of the formality of mathematics to represent a system as informal as a hospital directorate.

Can We Use Mathematics?

Related to the problem of reality is the problem of representation. We have chosen an extremely 'hard' representation medium with which to record our theory. The reason we have done this is twofold. Firstly the use of formal mathematics gives us great analytical power: we can deduce implications from the theory which help us to test its validity. Secondly, the logic and behaviour of the mathematics accords well with the logic and behaviour of a computer system which is the intended agent of change - the eventual purpose of the analysis.

Again however there are problems with this approach. We have seen how the 'reality' we are trying to model is a socially constructed one - one that is realised by the various groups of health care professionals. In so far as this reality resides in people's minds, it is not mathematical, and we should not expect it to have mathematical properties. That people do not think according to the rules of set theory is discussed convincingly by Lakoff [Lakoff87].

The difficulty manifests itself in two ways. Firstly the organisation might be considered to exhibit a number of behavioural properties which can be and are represented as formal predicates constraining the state space of the theory's models, but a theorem that is deducible from those properties might result in a behaviour that is not believed in or observed by the same people who related the original properties. The stated axioms, in the form of the theory's invariants, might be agreed with, and a theorem that is derivable from those disputed: mathematically the latter might be clearly and rigorously implied by the former, but unless the proof of this is very straightforward, the interviewee might not see why this is the case and remain unconvinced by the argument. Although the nature of this problem can be clearly understood, extended formal proofs were not used to any great extent in the derivation of the theory, and so the problem of disputed formal derivations did not arise (that the author was aware of).

Secondly a concept that is understood to have certain values and behaviours in one context can be considered by the same person to have different values and behaviours in another context. The nature of the concept is thus tailored by the individual concerned according to the situation she finds herself in. The concept of the bed comes in useful here again for didactic purposes. We saw before that a trolley could be perceived as a bed by some people but not by others. These different interpretations might be made by the

same person at different times. A doctor might be charged with looking after a number of patients in the Accident and Emergency department one of whom is lying on a trolley due to a lack of space in the admissions ward. The trolley would probably be perceived as a bed in this case, but if the same doctor then decided that the patient should be transferred to a particular ward where a 'proper' bed had come free, and that patient was wheeled to the ward on the trolley, she would consider it as a device for transportation - a more conventional understanding of what a trolley is. The two classifications might even not be exclusive - while the patient is using the trolley as a bed, the doctor might be planning how to get to the ward, and so be thinking of the trolley both as a bed and a transportation device simultaneously. This has as much to do with the social construction of reality as discussed in the previous section as with the problems of using discrete mathematics. However, the irrevocable placing of an element in a set is part and parcel of the use of this type of mathematics, and the variety and variability of classification is something not easily supported through the use of set theory[xxvi].

Of course, we can represent any observed behaviour using mathematics as long as we understand it. In the above example, 'bed' might be a property that could be applied to certain objects, of which 'trolleys' is a subset. Proceeding along this route, the theory can become arbitrarily complex and still not represent the 'truth' adequately. We cannot escape from the fact that mathematics can only approximate the constructed world, and not represent it entirely truthfully (We know this formally anyway from Gödel's theorem which tells us that in no formal system with the power of mathematics can we deduce all statements that are true in the system).

Conclusion

In conclusion we can say that there are a number of drawbacks that can be attributed to the use of the 'scientific method' and formalism, many of which have been encountered in the course of the project. We have seen that the conduct of refutative experiments in the form of structured interviews can be very difficult. This may be because the behaviours being examined are very slow, the 'correct' interpretation of an apparent counter-example might be unclear, the structured interviews being used as experiments are very difficult to control, or we have inadvertently stumbled into a 'paradigm trap'. The method does not really guide us when it comes to the construction of the theory which is (according to the method) arbitrary in scope and in terms of the categorisation of events into operations and the grouping of phenomena into either state or behaviour. The use of a unitary description means that it is particularly difficult to cope with the socially constructed nature of reality. This manifests itself in a number of ways: through the lack of co-extensivity of concepts, through different but equally valid descriptions of the same behaviour, and through the difficulty in identifying where either of these different aspects of reality construction have been encountered. We saw that the nature of discussion means that these problems, associated with different valid realities, will inevitably occur (though they can be minimised through the use of less abstract concepts in discussion). Finally we saw that the use of mathematics to investigate and describe such a soft domain as a hospital clinic leads to certain problems. The first of these is the occasional irrelevance or implausibility of logical derivation for interviewees, the second is the difficulty of assigning an object to a set when for any given domain worker, that object might take on different guises at different times, or even at the same time.

It would seem from the above section that there are a great number of drawbacks and limitations to the method described. We might be tempted to deduce that the approach was so flawed as to be worthless as a

[xxvi] Actually the representation of variable set membership is fairly straightforward if attributes are used to describe class membership. This approach can lead to excessively clumsy descriptions however, and is a way of 'working around' the limitations of the version of set theory used rather than recognising them and creating a mathematical structure that is in tune with the presentational framework.

means of deriving domain theories prior to the design of information systems. It might also seem strange to dwell so extensively on the problems associated with the method. It is only when we are sure of the weaknesses of a tool, however, that we can exploit its strengths and decide the appropriate manner to use it. In short, we should use the method only when we have successfully synthesised the understood strengths with corresponding limitations. Such a synthesis is presented below.

13.3.3 Synthesis

The aim of this section is to demonstrate that in spite of the problems discussed above, and although the method cannot provide us with an entirely reliable and indisputably complete representation of the domain we are concerned with, it can nevertheless give us valuable insights into the nature of that domain that might be overlooked if other methods were used. In a problem domain as complex and intractable as medicine, the use of the scientific method and formalism as described represents a powerful weapon in the armoury of the systems analyst.

The reason why we should not worry unduly about the problems discussed lies in the nature of the systems analysis and design problem. There are many methods, some of which are extremely well known, that we can use to produce information systems. In spite of these, the history of information systems development is littered with extremely expensive and costly disasters. Many of the problems that lead to the rejection of an information system can be traced back to an error in the initial analysis. Any method that helps us to trap a potential problem at the analysis stage must therefore be welcomed. When we have identified and rectified an error in the analysis, we cannot say that the new version is correct, but we can say that it is less flawed than it was. In this way, although some errors will slip through the analysis net as described in this thesis, some will be caught, and the quality of any resulting information system thus improved. Moreover, the method of domain analysis described is better at trapping certain classes of error than other techniques meaning that it is of genuine value in the information system design task.

Each of the problems discussed in the previous section places bounds on the usefulness of the method, but at the same time helps us see its advantages with greater contrast and confidence. We can address the issues raised above one by one and see how they should influence our understanding and use of the analysis conducted.

For example, we have seen how refutation can, for a number of reasons, be rendered extremely difficult. The fact that some refutations might be hard to find, however, should not stop us from looking, though we should be prepared for frustration. We should bear in mind that a refutation will be harder to find in behaviours that are infrequent or very slow. In the case of the theory as presented, this means that we should not be surprised to see that the majority of the refutations concern observed operational behaviour rather than the evolution (or possible values) of the 'configuration' of the clinic or even of clinics in general. If we look at the summaries of major refutations presented in the conclusions to Chapters 9 and 10, we will see that this is indeed the case. That there are few refutations for the long term behaviour of the clinic should not persuade us that the description presented is correct, and has been from the start, rather we should recall the difficulty of refutation in this area, and treat all claims that the theory makes with respect to the evolution of clinics with scepticism and care. In contrast to this, the profusion of refutations pertaining to the operational behaviour of the represented clinic should encourage us to use the part of the theory affected with greater confidence. In short, refutation is difficult, but at least it is possible, and its existence provides us with a powerful tool: an understanding of its limitations means that we can see where the tool is likely to have least effect thus helping us to avoid placing too much reliance on the theory in those areas.

The problem of the paradigm trap is an inevitable one that applies to all forms of human thought. Heidegger [Heidd62] recognised the phenomenon as concerning the way in which we perceive the world (Kuhn spoke about the problem specifically as it related to the development of empirical science): he explains that we are all blind most of the time to the most basic and common assumptions and properties of our lives. When a person runs to catch a bus, she is generally not conscious of exactly where she is putting her feet, how long her strides are, and how fast she is breathing: the action she is engaged in that is most immediate to her is 'running to catch a bus'. Heidegger tells us that the we can gain illumination by questioning our ideas and assumptions in this light (he suggests the use of language and etymology as a means to this end), but cautions that it is not a straightforward task. We are not only blind, but we do not know what form that blindness takes and what is being obscured from us. If this problem is as fundamental as Heidegger makes out, we should not expect the method espoused to escape it. However, we can get an idea when it might be especially useful or important to try and tackle this blindness and recast our understanding of the system being described. When the theory gets too complex for us easily to understand its main features and properties, then we should deduce that the particular viewpoint used might usefully be changed, and we should look about for another one. For example, the representation of the creator of an activity as another activity led to intolerable complexity in the theory, and a different way of understanding the behaviour of the domain was sought - a replacement of this paradigm with one where the creator of an activity is a health care professional has resulted in a far clearer conceptual structure. Similarly, the representation of all types of activity as essentially equal, be they as abstract as Diabetic Care or as specific as First Doctor Consultation, has led to a profusion of interacting attributes that endeavour to describe their differences. The complexity observed might encourage us to think of a different way of understanding the basic differences between activities of different abstractions. This is discussed further in Section 14.7.1.

We have seen that the scope of the theory is not guided by the method advocated. This too should not cause us too much concern. Where the scientific approach advocated is used, it should help identify problems that might have escaped us. Which is the most appropriate sub-domain to investigate and describe is up to the analyst. As a result it might be appropriate to use other methods and techniques to help guide the analysis. For example, Checkland's Soft Systems Methodology might help in identifying boundaries of the problem domain within which an information system might work. The author preferred to use a few explicitly stated guidelines (discussed in Chapter 7) and an educated guess at the sort of properties that might be significant in the development of an information system of the type described. The most basic scoping of a problem is beyond the ability of any rigorous method or problem structuring technique. Classical engineering can help us understand how to construct a bridge that will stand up - it will not tell us whether or not a bridge is required.

The most thorny problems with the use of a method such as that described is the constructed nature of reality - both in terms of its subjectivity and its ambiguity. Although this problem is intractable, certainly using a scientific approach, in a sense it does not concern us. The agent of change that we are hoping to introduce into the organisation is a computer system. This is a mathematical entity that works according to the laws of physics and manipulates symbols according to mathematical rules which it is designed (or programmed) to obey. One of the most fundamental of these rules is the law of the excluded middle. A proposition is either true or not true (though, due to the limits of computability we might not be able to discern which it is), or equally an element is either a member of a particular set or is not a member of that set. Two valid realities, one where proposition p is true or element e is in set S, and another where p is false and e is not in S, cannot both be represented by the information system at the same time. Somewhere a decision will have to be made as to which is the reality that will be supported by the information system. By using a unitary descriptive technique to conduct our domain analysis, we force such a decision to be

made consciously at the very earliest stages of the design process. In this way it can be discussed intelligently with the eventual stakeholders and users of the system without getting confused by such practical concerns as implementation data structures, efficiency of processing, cost of development and so on.

Many of the problems discussed in the previous section do not apply solely to the approach used in this project, but to all methods that rely on the use of description to record and communicate subjective ideas (such as the nature of reality). In many methods, the ultimate undecidability of reality is disguised by the ambiguity of the descriptive language used. By using a (formally) semantically rigorous notation to record our ideas, we confront the artifice of any representation all the more quickly and inevitably: an artifice that is inescapable if we are to proceed consciously and rationally.

That we can never really understand the problem is explained by Christopher Alexander in his book Notes on the Synthesis of Form [Alex71]. In this he contrasts two forms of design - self-conscious and unselfconscious. The primitive form of any design and build technique (the example used is that of the design and construction of dwellings - mud huts) is unselfconscious: the designer / builder does not make a conscious effort to relate the needs of the user of the designed artefact with its pertinent characteristics. She builds in the same way as her predecessors, perhaps making small modifications to cater for a specific new requirement (such as the need to build on a piece of ground that has not been used before). The fact that the changes to the basic design are small, and in response to limited changes of requirements, means that the nature of the designed artefact can gradually evolve along with the culture it is created to support. Alexander claims that this form of design and build is the most appropriate when the requirements change slowly with respect to the generation of the artefact to satisfy those requirements. In fact, in the work in question, Alexander portrays an almost Darwinian model of the evolution of artefacts. Due to the complexity of society and the things that are to support that society, and in particular the rich interaction of sub-parts of those things with each other and the supported society, it is in general impossible to predict the effects that a change in the design will have on the effectiveness of the eventual product (in Darwinian terms we could liken this to the pre-emptive opacity of the link between the genotype of an organism and its phenotype). If a change is made to support a new requirement resulting in an artefact that, due to an unforeseen interaction, does not provide a function that is required and was hitherto supported, the design will be 'selected against' and a new artefact will be designed and built which satisfies both (in the same way as a new genotype might find expression in a phenotype that is ill suited to its environment resulting in its 'de-selection').

While this form of unselfconscious design, and its proto-Darwinian evolution, is the best method of getting a close fit between designed artefacts and the requirements of the society where the needs of that society are changing very slowly, it is not suitable when the needs of the society are changing quickly when compared with the 'generation span' of the product. This is the case in much of modern society - our 'need' for cars, buildings and information systems changes too quickly for the evolutionary approach alone to be feasible. In this case we must rely on self-conscious design where we try and predict in advance the nature of the thing we are designing, and how it will satisfy the requirements we are trying to support. In its turn, self-conscious design must rest on the use of description to communicate with others, and for communicating with ourselves over time. In short, although the use of description in design is fraught with problems, for complex products that do not have time to evolve unselfconsciously, it is a necessary evil. Having accepted the need for description, the use of formality gives us some benefits, and removes some of the superficial ambiguity associated with other notations. In other words, all the alternative approaches share the same underlying problems and have others besides.

The design and building of information systems is clearly an area where, from the argument presented above, we must rely on self-conscious design - the requirements of such an artefact are extremely complex, and its generational span far too long to rely on the natural evolution of a system. Once we have admitted that the problem is one of great difficulty, that has never been satisfactorily resolved (and probably never can be), then we are in a position to recognise that the method described, with its attendant scientific method and formalism, gives us an additional and powerful perspective over it. We should not imagine that the insight it gives is complete or that it is superior to that gained from any other perspective. Indeed, we should take heed of Morgan's work, Images of Organisation [Morgan86], and recognise that, especially when it comes to entities as complex as human organisations, many ways of interpreting what is observed should be encouraged, and that wisdom does not come from slavishly following one intellectual ideology. Within the narrow and mechanistic framework that has been prescribed, the method used represents a powerful and consistent technique that could usefully be added to the analyst's armoury.

One last comment ought to be made while we are still considering methods for describing reality. Many of the drawbacks in the method are mitigated somewhat because the agent of change we are intending to introduce suffers from them also. The fact that discrete mathematics has been used to describe reality presents less of a problem as computer systems are bound by the same formalism - if we accept that a computer system is going to be introduced, we have already resigned ourselves to the limitations of formality and so should not worry unduly that the descriptive method used shares the same weaknesses. We should be more cautious however when it comes to using the approach to design systems that do not have mathematical underpinnings. For example, if we are intending to re-design the business processes of the organisation for some defined end, the eventual system that will be implemented (the new business process) will certainly not be bound by the strictures of set theory and the predicate calculus, and any reasoning mechanism that assumes it will be will inevitably be flawed. This does not mean that the use of formality should be avoided in this area, and indeed novel and exciting work has been done in this very area [Glykas94]: however, great care should be taken and the limitations of mathematics recognised.

13.4 Issues concerning Construction and Analysis of the Information System Specification

13.4.1 Introduction

The stage of development which requires the most involved thinking to understand, and for which the objections are most profound and far reaching is the development of the domain theory, and so we would expect the previous section to be more far-reaching in its analysis of the techniques espoused. The purpose of the project was not to produce a description of the organisation, but rather specifications of information systems components that will support that organisation. The step from domain description to information system specification is thus central to the purpose of the work, and we must investigate the arguments behind the techniques involved in taking the step.

This section therefore explores the second step in the systems design process described in Section 13.1. The justification re-iterates the method used to construct the information system specification through the use of an appropriate interaction theory, and presents a computer science interpretation for the four 'motives' that guide this process. The criticism questions each of the motives and describes the shortfalls of the approach. The synthesis likens the interaction theory to a map - it will not prevent us from making mistakes, but will mean that we will act in a more informed manner when making decisions.

13.4.2 Justification

The Interaction Theory

One of the unusual aspects of the method espoused in this thesis is the use of an explicit 'interaction theory' which links the information system specification with the domain theory. The intended interpretation of the domain theory is as a selection of rules governing the behaviour of entities in the world. As we have seen, we can (to a greater or lesser extent) examine the feasibility of this theory by comparing its predictions with the behavioural scope of the real organisation as perceived by domain participants. In a similar way, the intended interpretation of the information system specification is as a selection of rules governing the behaviour of data files in a computer system. We can (as we shall see) examine the accuracy of the specification by investigating the properties that the real information system possesses (once it has been built) and comparing them with those claimed by the specification. We should not directly interpret the components of the specification as entities in the domain as this is the role of the domain theory. However, we know that we want the implemented information system to be interpretable by a domain worker as a model of the domain, so clearly we have to find some method of determining the adequacy of an ir formation system built to the specifications given. This is the purpose of the interaction theory. This is a formal theory that records a set of interpretations that we would like domain workers to make on information system components in the implemented system. For example we might want a data file labelled 'Patients' to be interpreted as a set of real patients in the domain. We can talk about characteristic behaviours of the set of patients in the domain, and about behaviours of the data file labelled 'Patients' in the information system, but without the interaction theory we have no formal way of seeing if, in an implementation of the specification, the data file called 'Patients' is a good representation of the real entity. By introducing an interpretation function (and conversely, a representation function), we can see whether an implemented data set shares the same properties as the entity we wish it to represent, and where any shortcomings might lie. In the case of the set of patients, we have the interpretation function IntP where

dis-int-T 6: $IntP$: $crs\text{-}Pid \longrightarrow Patients$

We now have a formal means of talking about the information system in terms of the domain. We saw in Chapter 11 how we can use such a construction to investigate the limitations of existing information systems. Such limitations might lie in inadequacies of scope where state components in the domain theory are only partly or not at all represented, limitations of functional support where operations that take place in the domain are only partly or not at all supported by the information system, and errors where a consistent interpretation is not possible for a particular data file.

Design of IS

Using the interaction theory, we can get an understanding of the degree to which an existing information system can be though of as a representation of the domain. Perhaps more important however, is the design of new information systems.

It is clear that a new information system need not share all the 'inadequacies' of those that already exist. In the limit, we might choose to implement a model of the domain theory itself. If we did this then (to the extent of the adequacy of the domain theory) the information system would be a 'perfect' representation of the organisation, and a complete and consistent interpretation would be possible. There are several reasons why this might be undesirable.

Firstly, we need to ensure that the implemented system is as simple as possible. The more complex the data structures and processes of the finished information system, the more likely we are to discover

unexpected errors. Similarly a complex information system is generally more difficult to maintain than a simple one. If the information system is not a good representation of the domain of interest, then at least the user should be able to understand the underlying logic and hypothesise a domain that the information system would support. If this is possible then the user can consciously derive utility from the system in spite of its inadequacies. Additional complexity means that any erroneous behaviour becomes very difficult to understand meaning that the information system is correspondingly less useful. In short, when it comes to information systems there is a certain virtue in simplicity.

Several of the operations in the domain are characterised by many arguments - a degree of detail being necessary to describe the change of state with sufficient completeness. If we want to represent the state of the domain completely we will thus require the user to provide the information system with a great deal of data - adding to his or her workload and inevitably contributing to user hostility. Any reduction in the number of parameters we require of the user will have to be accompanied by a reduction of verisimilitude of representation of the domain, however.

Not only do some operations require a more work to represent than they are worth, but some operations we do not want to represent at all. For example, when a patient is admitted to a ward with acute hypoglycaemia, she needs to be treated and her condition stabilised, not recorded on any computer system. Other operations might record behaviours that we are totally unconcerned about, and that do not interact with an 'important' part of the domain in any significant way. Here again, the function we are supporting would be superfluous to the needs of the eventual system users.

Finally, we need to consider existing technology. In most system development projects there will be existing, or legacy, information systems that need to be integrated to form a single coherent system. The desirable change to these systems is also the minimum change - we should make the 'wrapping' software which binds the legacy systems together as simple as is reasonable. Clearly in this case the nature of the existing systems will influence the structure of the final one. Legacy systems are one form of 'technical artefact' that we need to consider when designing the new integrated system. The implementation language is another. Different languages or application development environments treat different data structures and behavioural forms with greater or lesser efficiency. For example, the new departmental clinical record system and its extensions are being implemented using a fourth generation language and a relational database. Using this technology, simple sets and relations are easy to implement whereas more complex structures such as directed acyclic graphs, lists, and interacting complexes of relations are more difficult to implement. It is sensible to implement structures that can be easily constructed from these primitives of the language or environment that is being used.

To summarise the above paragraphs, in constructing the information system we are encouraged to keep it simple - simple to construct, simple to maintain, and simple to use. The simplest approach is clearly the 'no change' option, or just to continue using the legacy systems already in place. Clearly this is not going to enable any change for the better - there must be some form of compromise between extreme simplicity and total representation of the domain. The choice of the position of this happy medium can only be determined on the strengths of the case being considered in consultation with users of and stakeholders in the system. However, there are a number of areas where conformance with the domain theory is more important than others. The author has grouped some of these into four guidelines to be used to help us decide which new computer support is especially beneficial. These have been called developmental motives as they provide the counteracting force to the drive for simplicity behind the decisions made when constructing the specification of the integrated system.

The four developmental motives have already been discussed at great length in Sections 12.4. We will revisit them briefly here and examine them from a perspective that should give them greater clarity and place them in the context of mainstream computer science. This context is that of specification reification.

Four Developmental Motives - Variations on the Theme of Reification

Reification is a term that has been coined by H. Zemanek to describe the process which takes an abstract formal specification of a system and alters its form so that it can be easily implemented on a computer while preserving the essence of the behaviour described in the specification. Jones says:

'The style of formal specification ... uses abstract models of data types... High-level design decisions normally involve choosing the representation of data: *data reification* involves the transition from abstract to concrete data types and the justification of the transition' ([Jones90] pp180).

The justification of a reification step hinges on the adequacy of its *retrieve* function. The retrieve function returns an abstract (pre-reification) state when supplied with a concrete (post-reification) state. Three points should be made about this function. firstly it is a function rather than a relation, secondly it is complete, and thirdly it covers its range - it is surjective. If the abstract specification truly is abstract, then we can assume that each different state must be represented by models of the concrete specification. If retrieve were a partial function, then there would be states of models of the concrete specification that represented no possible states of a model of the abstract specification: behaviours that were explicitly forbidden in the abstract specification might be permitted in the concrete specification which is clearly undesirable. If retrieve did not cover its range then there would be states of models of the abstract specification that were not represented in models of the concrete specification and a computer system implemented from the concrete specification would not fully support all the functions originally specified. If retrieve were a relation rather than a function then we could not in general determine which of two abstract states were being represented by a concrete state - any decision that we took would again lead to incomplete representation of models of the abstract specification. Note that the retrieve function is not an injection - it is not necessary for each concrete state to be retrieved to a different abstract state. We can see that this would be the case if we consider the representation of a set in the abstract specification as an indexed table in the concrete specification. The abstract data type does not distinguish between different ordering of its elements whereas the concrete one does. As long as each member of the (abstract) set has a single corresponding member of the (concrete) indexed table, then any ordering of the latter would represent the same set. In this case if there were n elements in the set, then there would be $n!$ possible representations that could be retrieved to only that set.

Jones defines a retrieve relation for each data type in the concrete specification that is to represent an abstract data type: a number of distinct retrieve functions (each with a different name) are needed to reconstruct an abstract state from any concrete state. For the sake of the argument presented here, we will not worry about the interaction between the different retrieve functions as they apply to a particular state component, but rather talk about that functional construction, which can be created from all the separate retrieves, that returns a valid state of the entire abstract model from a state of the entire concrete model. We will talk only of this constructed function from now on, calling it *retrieve*.

We can represent the arguments presented above in graphical form as follows:

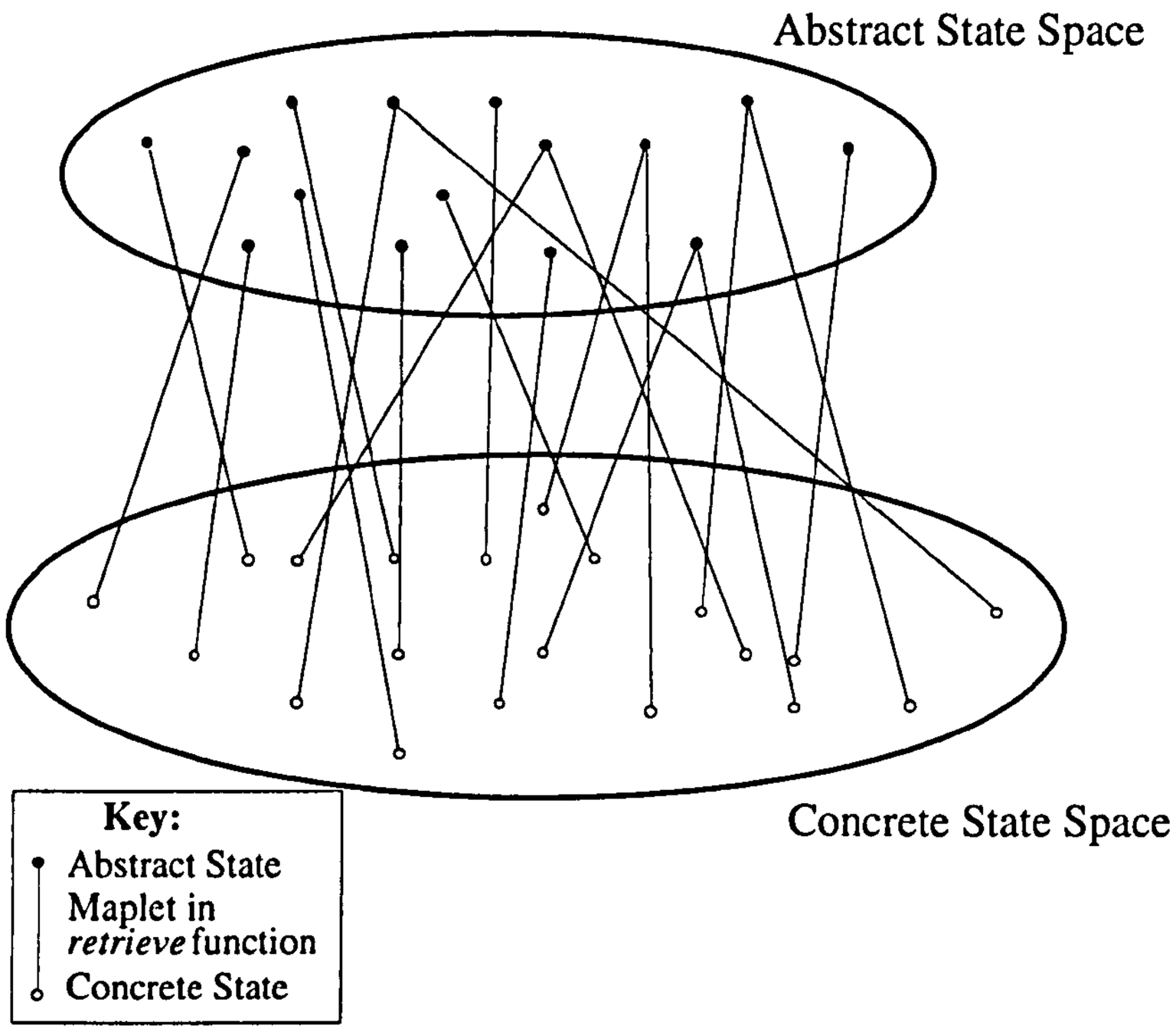


Figure 4-2: Concrete and abstract state spaces

The ellipsoid drawn at the top of the figure represents the state space of a model of the abstract specification. The ellipsoid at the bottom represents the state space of a model of the concrete specification. Each state in the concrete state space is mapped to a unique state in the abstract state space by the appropriate *retrieve* functions. Notice that *retrieve* is functional (each concrete state is mapped to only one abstract state), total (every concrete state is mapped to an abstract state), and surjective (for every abstract state there is at least one concrete state that maps to it).

We will be discussing different mappings from one space to another in some detail below, and will be making of figures illustrating points similar to those illustrated above. For convenience, we will use a 'shorthand' version of the above illustration. One such is presented below:

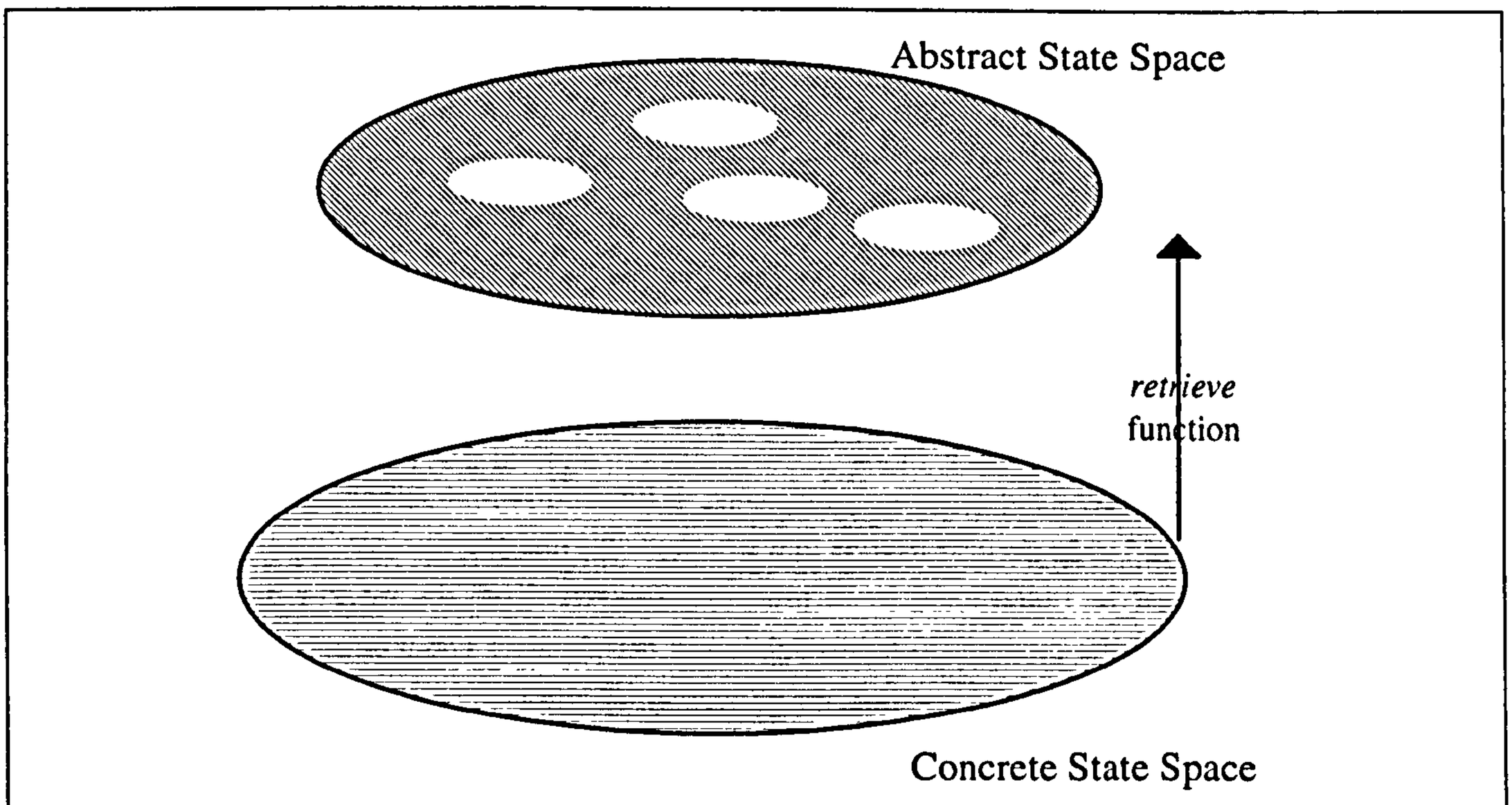


Figure 4-3: Simplified diagram illustrating an argument about state space mappings.

In the figure above, the lower ellipsoid represents the state space of models of the concrete specification while the one above represents the state space of models of the abstract specification. The shaded area in the abstract ellipsoid represents the range of the *retrieve* function. In this example the retrieve function is seen to be inadequate as there are abstract states (represented by the contents of the smaller ellipsoids) that are not represented by any concrete state. An accurate 'graph' representing the retrieve function would have to be multi-dimensional, with a separate dimension for each distinct state component. If a concrete state component was not retrievable onto the abstract state space in any way, then the abstract state space would have a lesser dimension than the concrete one. This is obviously difficult to draw and so has not been attempted. Suffice it to say that diagrams which attempt to show, for example, increased scope of coverage in the abstract state space (as here) should be understood as not only having greater (multi-dimensional) volume, but also possibly greater dimensionality.

The rules governing adequacy of the *retrieve* functions are just that - they are not guidelines and are not supposed to be enforced loosely. If the retrieve function can be shown to be non-functional, partial or non-surjective then the concrete specification is a faulty reification of the abstract specification. This demonstration can be performed according to the laws of logic and set theory - that is formally. It is the ability rigorously to demonstrate the adequacy of retrieve functions that makes formal methods so powerful - we can not only specify an abstract system, but we can show that a particular computer program is a valid implementation of that specification.

There are many similarities between the reification of an abstract specification to form a concrete one and the derivation of information system specifications from a domain theory. Instead of a number of *retrieve* functions to take us from concrete to abstract state, we *interpretation* functions to take us from specification to domain theory. As in the case of reification, we are taking an abstract representation of a system and contriving to manipulate it so that it takes the form of some more readily implementable structure. The differences between reification and specification derivation cover more than just the names

of the different concepts however: when it comes to deriving an information system specification from the domain theory, the rules are less hard and fast. As pointed out in the previous section, there may be good reasons why a perfect implementation of the domain theory is not desirable. However, the rules for reification of abstract specifications can be 'diluted' to form guidelines, or motives, for the development of computer systems. Each motive can be seen as a derivative of the three rules that adequate *retrieve* functions must obey.

The four motives are: expansion of scope, functionality of interpretation, reduction of entropy and restriction of prohibition. In the next four sections, each of these motives will be discussed with reference to one or more of the retrieve function adequacy rules.

Expansion of Scope

The first motive is the gratuitous extension of the scope of the information system. The reification rule that this can be considered to be derived from is that which insists that the *retrieve* function is surjective. Every abstract state must be represented by at least one concrete state. As we saw earlier, there are perfectly good reasons for the incomplete representation of certain state components in the information system, or for omission of other state components altogether. However, in the absence of such reasons, and where the increase in complexity of the system is supportable, we should take the opportunity to expand the scope of the information system, both in terms of increasing the coverage of represented state components and of representing previously unsupported state components. The more of the domain that can be understood through interpretation of the information system the better as we will end up with a richer and more 'realistic' picture of that domain from an implemented information system. Clearly the greater the scope of coverage obtained by projecting the information system state space onto the domain state space through the interpretation function, the more closely that function is to being a surjection. We are thus justified in comparing this motive with the reification test that insists that *retrieve* must be surjective.

The argument presented here is illustrated by the figure below:

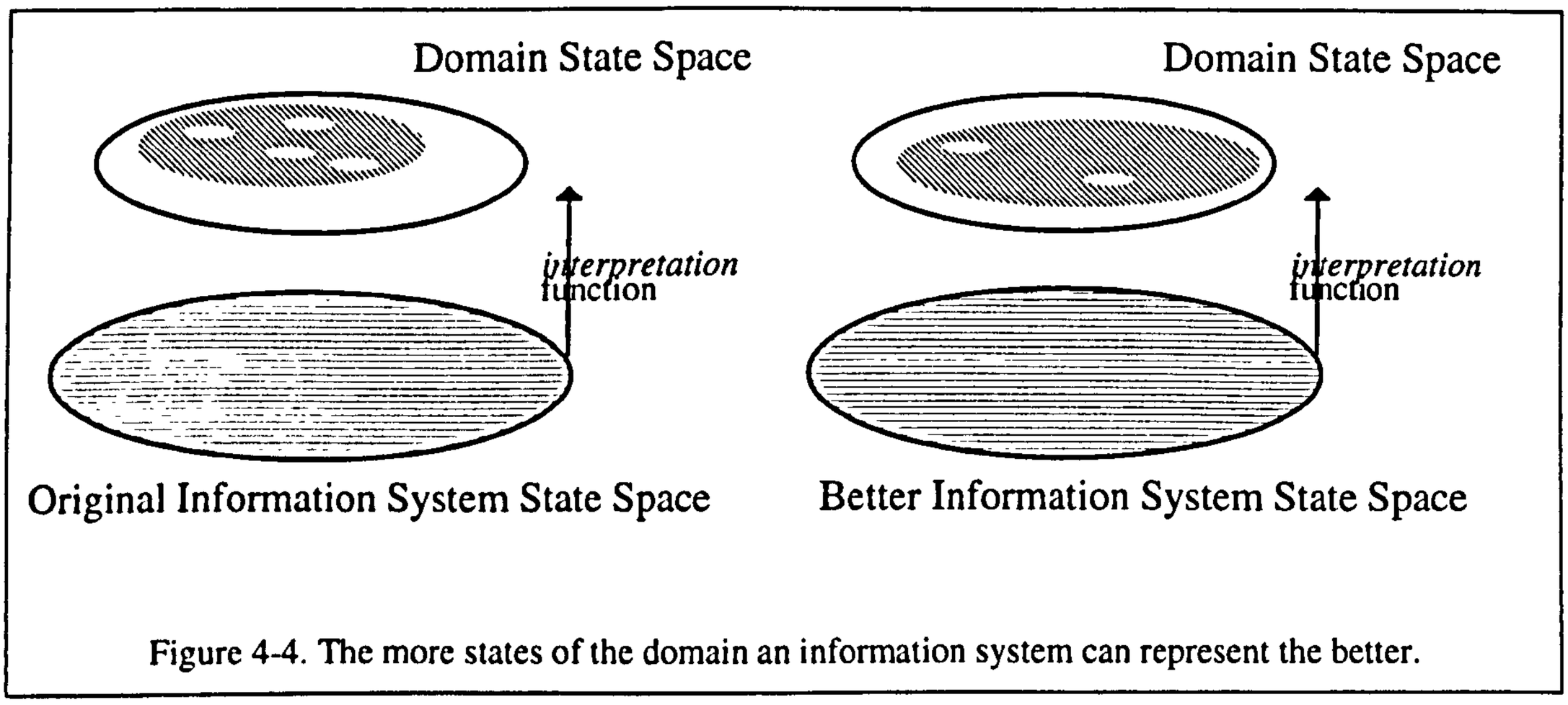


Figure 4-4. The more states of the domain an information system can represent the better.

Of the two information systems, the one on the left has a smaller coverage of the domain state space than the one on the right. For this reason, the interpretation function for the information system on the right is 'closer' to an injection than that of the information system on the left and so the information system can be said to be 'better'. Of course, these 'improvements' would have to be weighed against the problems caused by any increase in complexity. If these problems are negligible, then we can say that the scope extension is gratuitous and so should be encouraged.

Functionality of Interpretation

The second developmental motive is the functionality of interpretation. This requires some care in its definition. Loosely speaking we could say that this motive leads us to create information systems where every state of that system can be interpreted as one and only one state of the domain. The reification rule this is based on is that which insists that *retrieve* is functional. Every valid concrete state should be able to be mapped onto a valid abstract state. If we look at the information systems described above, we can see that this is generally the case. For example, if a patient is recorded on the system as being registered, then we should interpret this as meaning only that the patient has indeed been registered, and not allow ourselves an alternative valid interpretation that the patient is not registered. The argument behind this insistence should be clear. We do not want two conflicting states of the organisation to be described by the same values of the database: if we allow this, then decisions taken on the basis of information form the computer system will have to bear in mind that two or more, possibly very different, situations might hold in reality and the user would have to be sure that she understood the implications of the decision in these different possible worlds. In short ambiguity, although not rendering the information in question useless, drastically reduces its value.

The reason why we must use care is that in an incomplete representation (and we saw earlier that any representation is always incomplete if not of the finite domain theory then certainly of the essentially infinite domain), there will always be some ambiguity. For example, none of the information systems specified records at any stage the identity or profession of the people running the clinical activities described. If at any time the current state of the information system described an Initial Doctor Consultation as proceeding, a number of different valid interpretations could be made - one for each of the valid health care professionals capable of running that activity. Similarly, in the initial Clinical Record System, no mention is made of appointment slots. In this absence of information about slots, an activity could be understood as having been booked for any time in the future implying an effectively infinite ambiguity of interpretation. We are thus asking far too much of the information system if we insist on functionality of interpretation of the system's state. If it is to be useful, this motive has to be qualified.

What we in fact require is that any state of the information system can be interpreted as only one state of that part of the domain theory covered by the system. That part of the domain theory covered by the system is defined as the interpreted values of all possible system states - that is the range of the interpretation function. Thus all we have said is that our interpretation must be functional. The care comes with the observation that the function will not be totally surjective over the state space of the domain theory: there will be valid states of the domain theory that the information system does not totally describe and yet have interpretations of all the (interpretable) components in the information system. This argument is illustrated in the following diagrams:

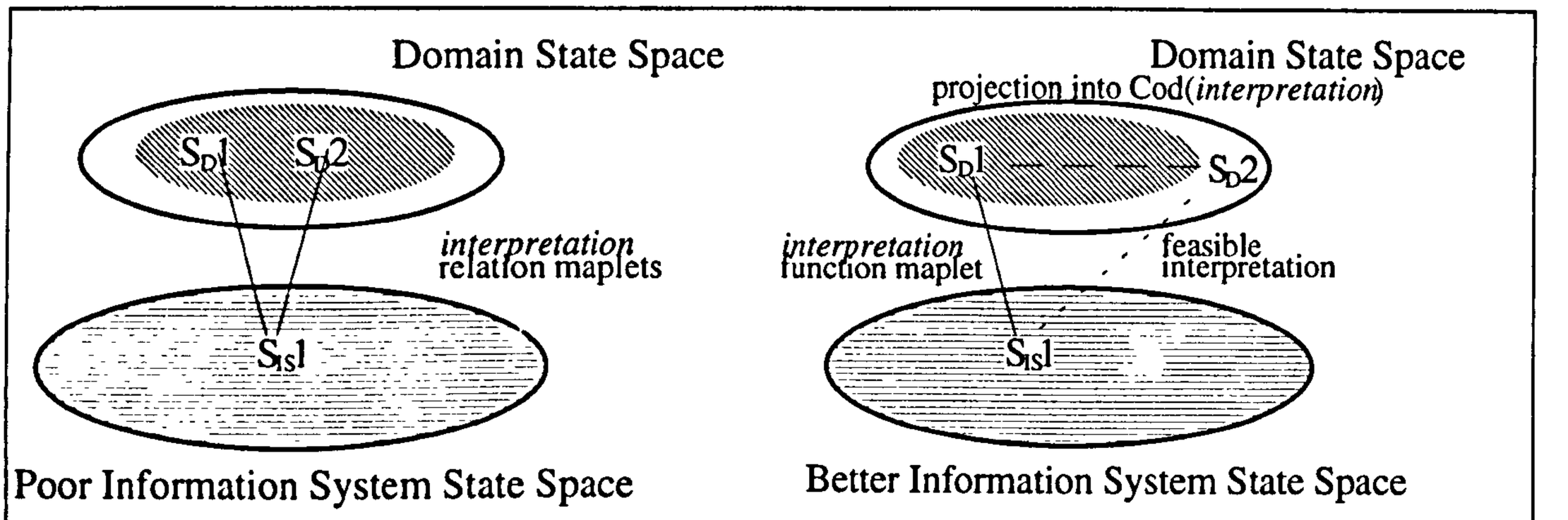


Figure 4-5: Illustration of the motive encouraging us to look for functionality of interpretation.

Note that in the first diagram S_D1 cannot be a projection of S_D2 and *vice versa*.

In the above figure, the diagram on the left illustrates a poor information system. The single information system state $S_{IS}1$ can be interpreted as one of two possible distinct domain states: S_D1 and S_D2, both of which are in the range of *interpretation* which is thus in this case a relation. In the diagram on the left a somewhat different case involving a better information system is presented. In this case, $S_{IS}1$ has only one *interpretation* - that is, *interpretation* is a function though not surjective. The domain state S_D2 represents a feasible interpretation that is not specifically represented by $S_{IS}1$. The reason why S_D2 is a feasible interpretation is that it can be projected onto S_D1 which is the direct interpretation of $S_{IS}1$. A projection involves the compression of dimensions - in this case dimensions that are not represented by the information system. Thus states in the domain space can be projected onto a state (which may not be permissible in the full domain space) with the same dimensionality as the (interpretable) information system.

For example, suppose we take the state of a domain which gave the health care professionals (if any) running each proceeding activity. The projection of such a state into the range of the *interpretation* function would involve simply removing all mention of the health care professionals from the state in question, along with any other parts of state components that are not represented in the range of *interpretation* - activities for example. Of course, the removal of the health care professionals (and any un-represented activities or other values) might mean that the state no longer lies within the legitimate domain space: some activities need to have health care professionals associated with them to run at all. This is allowable as long as the states that have been projected <u>are</u> valid. The idea of projection to reduce the dimensionality of a value in a generalised space (normally called a vector space) is a standard mathematical technique. What is slightly different here is that we are not only reducing the dimension of the space but also paring away those values that are not supported by the information system. This extra restriction on the values of the projection are necessary as we allow many of the represented concepts to be subsets of the concepts as they exist in the domain. For example we said that

crs-int-T 5: $IntP$: $crs\text{-}Pid \rightarrow Patients$

from which we can deduce that

$Cod(IntP) \subseteq Patients$

and

crs-int-I 6: $Cod(IntV) \subseteq Activities \backslash Out$.

In short, for the purposes described above, a state S_P, can be though of as a projection of another S_O, if S_P can be created only through the removal of values from S_O.

Obviously we can say nothing about the state of activities that are not in Cod(IntV) from looking at the information system, and so these must be excluded from the state space we must construct to judge whether the *interpretation* function is indeed functional.

Although the concepts presented here may seem complex, they are based on a fairly straightforward idea - that interpretation must be functional. The complication comes when we observe that the function is not surjective and that great care must be taken when deciding whether a given specification has been designed well according to this motive.

Reduction of Entropy

The third developmental motive is the reduction of entropy. The reification rule this is derived from is that which requires the *retrieve* function to be complete. What we might say in this case is that all states of the information system can be mapped onto valid domain states via the *interpretation* function. This simplistic understanding can be justified in the same way as we defended the decision to increase the 'refutability' of the domain theory: the greater the constraints on the world imposed by the theory, the more we know and understand about the world (from the theory). If the information system supports states that can be interpreted as illegal domain values then clearly some of the constraints represented in the domain theory have not been introduced into its representation, and so the weaker and poorer that representation is (in the limit such an information system might take the form of a word processor or even simple text editor). As with the last motive, however, the situation is rather more complex than this in a number of ways.

Firstly, even when two states are both capable of being mapped onto valid domain states, the information system might still be inadequate with respect to them. The reason is that we must not only ensure that all the states of the information system can be mapped onto the domain, but also all the behaviours as well. If the information system allows an operation that alters its values from expressing one valid state to another, we must be sure that the domain theory allows an operation that moves the domain values from one interpreted state to the other. The argument behind this motive has been discussed at length in Section 12.4.5. Put briefly, the exclusion of representations of invalid behaviours prevents the user being presented with 'nonsensical' operations to choose from which would reduce the usability and usefulness of the system.

These arguments can be illustrated with the appropriate diagrams. Firstly we can see what the 'ideal' situation might be. This is the purpose of the following figure.

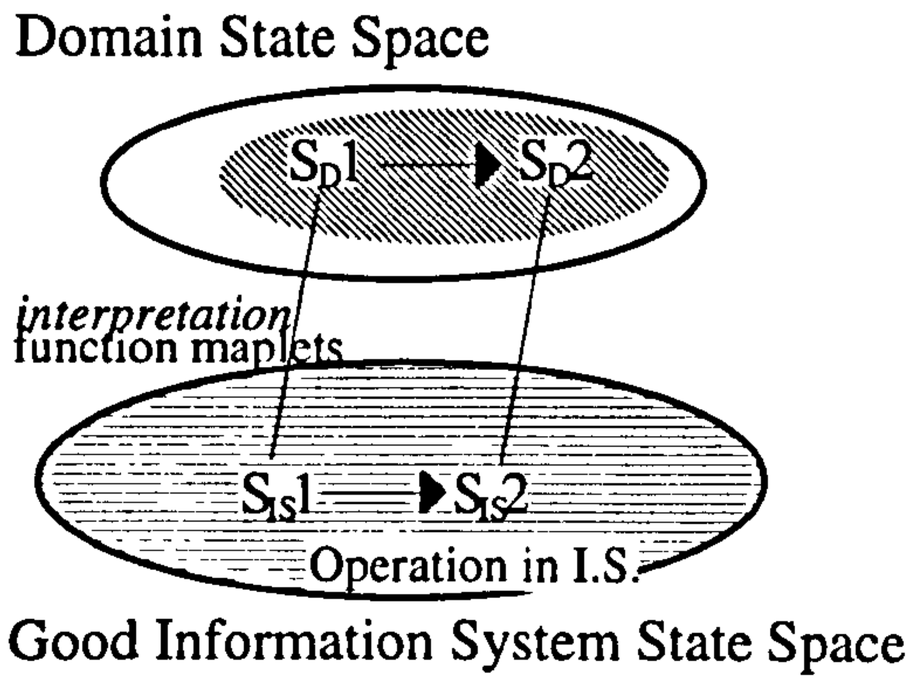


Figure 4-6. A good information system where entropy of information system is minimised.

Here both the states and the behaviours of the information system can be interpreted as valid states and behaviours in the domain.

We can then look at two cases where the information system might be judged lacking with respect to the domain theory.

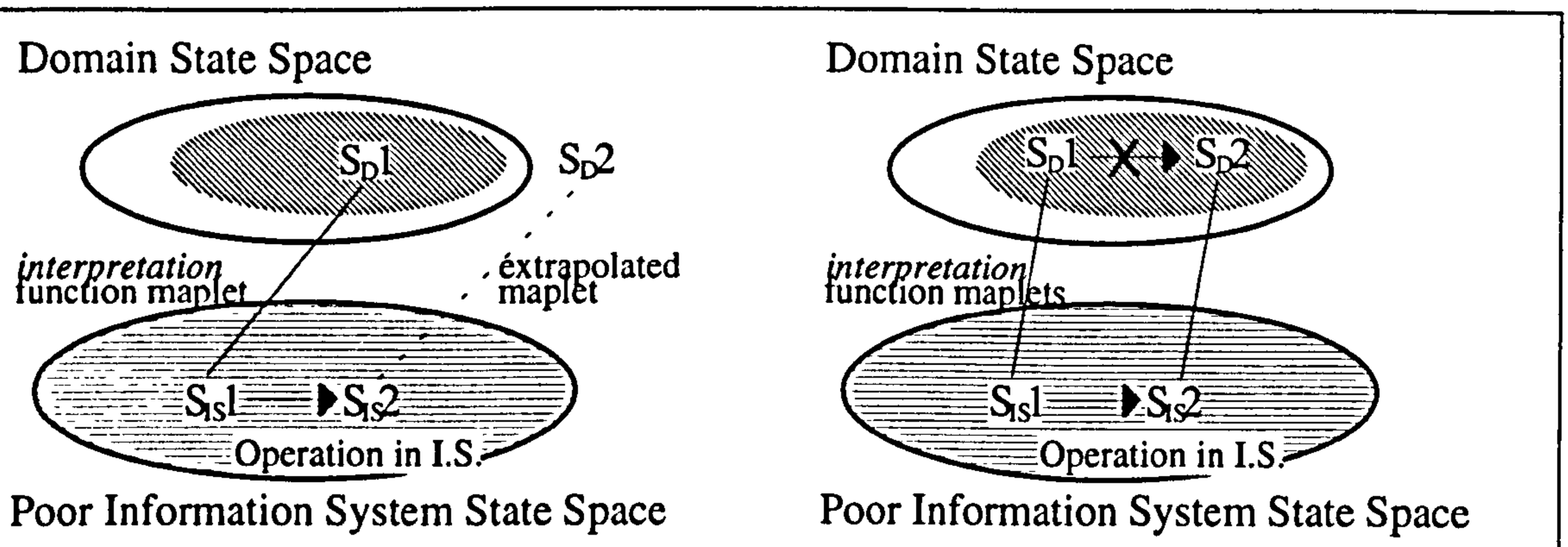


Figure 4-7: Poor information systems judged by the motive described here.

The diagram on the left illustrates the case where there is a state of the information system, $S_{IS}2$ that would be mapped by *interpretation* (if such an *interpretation* were possible) onto a state that is illegal in that it violates laws stated in the domain theory. The diagram on the right shows the case where two states of the information system, $S_{IS}1$ and $S_{IS}2$, can be mapped through interpretation onto valid states of models of the domain theory: S_D1 and S_D2. Although the states are thus permitted, the information system allows a behaviour which has $S_{IS}1$ as its pre-state and $S_{IS}2$ as its post state: the domain theory allows no behaviour which takes S_D1 as its pre-state and S_D2 as is post state. The user of this system would thus be presented with a 'nonsensical' operation which is undesirable and reduces its usability.

There is a subtle addition to this motive that needs some careful thought. This is that the information system might support a behaviour that is not illegal in the projection of the domain theory state space that is the range of the *interpretation* function but is nevertheless forbidden by the full (unprojected) domain theory. The way to understand if this is the case is to take a state from the state space of the full domain theory, which we shall call S_F1, and see if it can be projected onto the range of the *interpretation* function, as a state which shall be called S_D1, which is in its turn an interpretation of a state of the information system, $S_{IS}1$. What we would like to be able to say now is that every post state of S_D1 which

is an interpretation of a post state of the information system (where the pre-state is $S_{IS}1$) is a projection of a possible post state in the full domain state space where the pre-state is S_F1.

This somewhat complex notion can be better explained with a diagram.

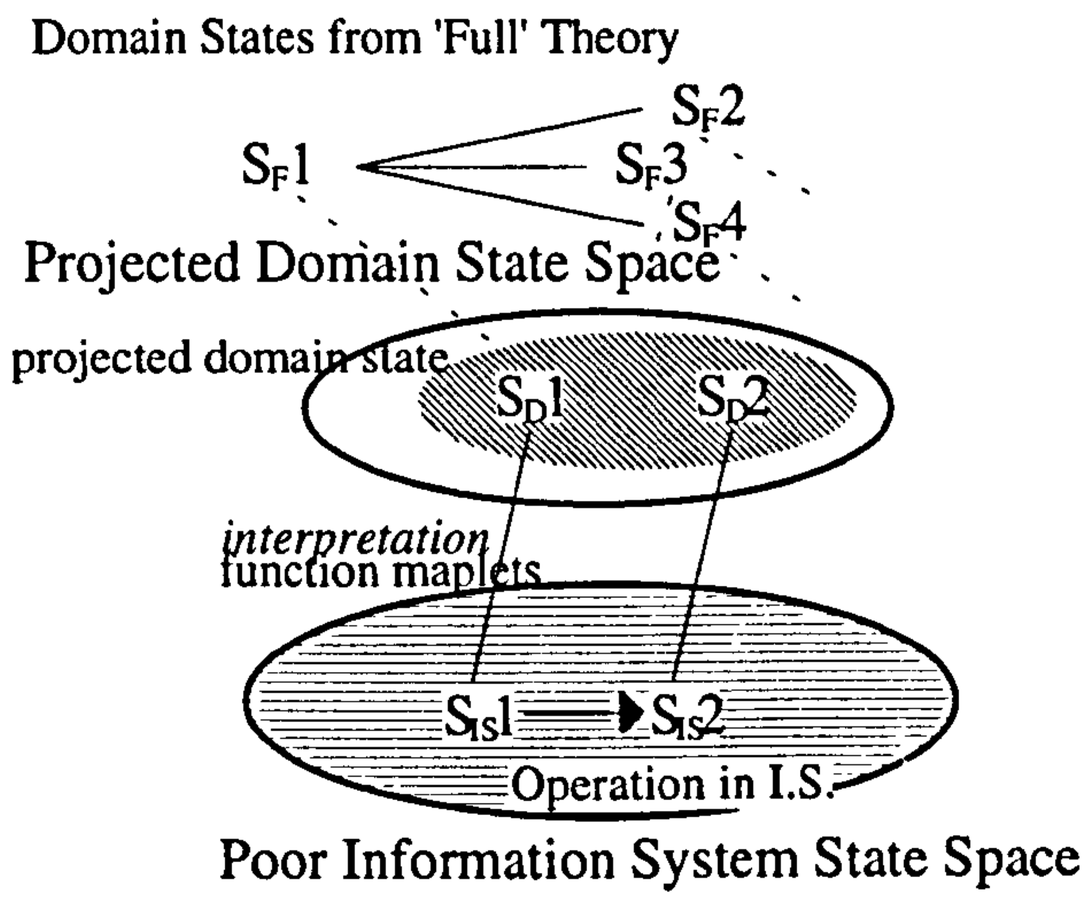


Figure 4-8: Information system that allows operations that are illegal in the 'full' domain state space.

Note that none of the possible states reachable for 'full' domain state S_F1 (that is, states S_F2, S_F3, and S_F4) is capable of being projected onto S_D2, the interpretation of $S_{IS}2$. Thus the operation that takes us (in the information system) from $S_{IS}1$ to $S_{IS}2$ is 'non-sensical' and should be prevented if (reasonably) possible.

In the above diagram, note that none of the possible states reachable for 'full' domain state S_F1 (that is, states S_F2, S_F3, and S_F4) is capable of being projected onto S_D2, the interpretation of $S_{IS}2$. Thus the operation that takes us (in the information system) from $S_{IS}1$ to $S_{IS}2$ is 'non-sensical' and should be prevented if (reasonably) possible.

In general it will not be the case that with the existing state components in the information system that we will be able to prevent this sort of illegality occurring. However, we should be aware of its existence, and if the opportunity presents itself to easily introduce an additional state component into the information system that enable us to capture and prevent these invalid operations, then we should seize it. The decision to expand the coverage of crs-VisPid from representations of proceeding and complete activities to representations of all activities (ie, requests as well) is one of these cases.

This more subtle variation of the entropy reducing motive can be summarised as follows. The more a new component would (validly) constrain the behaviour of the information system, the keener we should be to introduce it.

Restriction of Operations

The final motive has been called here the prevention of prohibition. What this cryptic name is intended to convey is that we do not want the information system to prevent us from carrying out a (useful) procedure that was previously supported in the organisation. This is slightly different from the other motives described in that it requires some form of statement of the intended use of the information system. We can easily understand the times when our design decisions might be modified by this motive as these are essentially the reverse of the cases described in the above section on the reduction of entropy. There we wanted to prevent the information system from providing operations that were illegal in the domain: here

we want to ensure that the information system provides all the operations that are legal in a specified part of the domain. This motive is thus similar to the first in that it encourages expansion of scope of coverage of the domain by the information system and thus parallels the reification rule which insists that the *retrieve* function must be surjective. The motive in question is stronger however in that we do not want to increase the coverage provided by the information system in a general sense, we want to ensure that it is total within a certain specified area. This reflects the case when the information system provides all the support that is available for a certain class of behaviours, and so the coverage of the domain by the information system is *de facto* complete. The motive tells us that in constructing the information system, we must ensure that within this area, there are no behaviours of the domain that are prevented by our insistence that the coverage of the domain is complete.

The area where the coverage is to be complete must be specified in the interaction theory as this is the only place that we can talk about the state components of both the domain and the information system. By constructing invariants over these state components from both theories, we can ensure that at any time there is a complete representation of the sub-domain in question in the information system. For example, we might say that for certain activity types, all activities of those types must be supported by the information system. This might be departmental policy - in the DEDC the policy is for every Initial Dr Consultation and Followup Dr Consultation to be recorded on the computer system at the consultation in question. In the interaction theory we could record this by saying that at any time there are no proceeding activities of the relevant types that are not in the range of the appropriate (local rather than global) *interpretation* function. This might be specified formally as

$$\text{dis-int-I 20} \quad (im\ ActType^{-1})\ FullRepTypes \cap Proceed \subseteq (im\ IntA)\ crs\text{-}Proceed$$

where *FullRepTypes* is the set of types that is to be totally supported by the information system. What we have done here is define an intended usage of the information system - we cannot enforce these invariants through skilled design of the information system, or in any (socially acceptable) manner at all. What we can do is see what effect on the domain - information system ensemble such an intended usage would have. Thus, having specified the area where support or coverage is to be complete, we are obliged to investigate the behavioural implications of this decision. In particular we want to determine if the provision of this compulsory support prevents certain behaviours that might be observed in the domain as it is without any information system. An obvious example of this can be imagined in the case above: if all activities of a certain type need to be recorded on the information system, and the information system doesn't in fact recognise activities of one of those types, then no activities of that type can run at all. In general the restrictions that the intended use of an information system places on the domain will be more subtle than this, but the effect might be equally unacceptable, and we should at least know about them. This argument is summarised in the following figure:

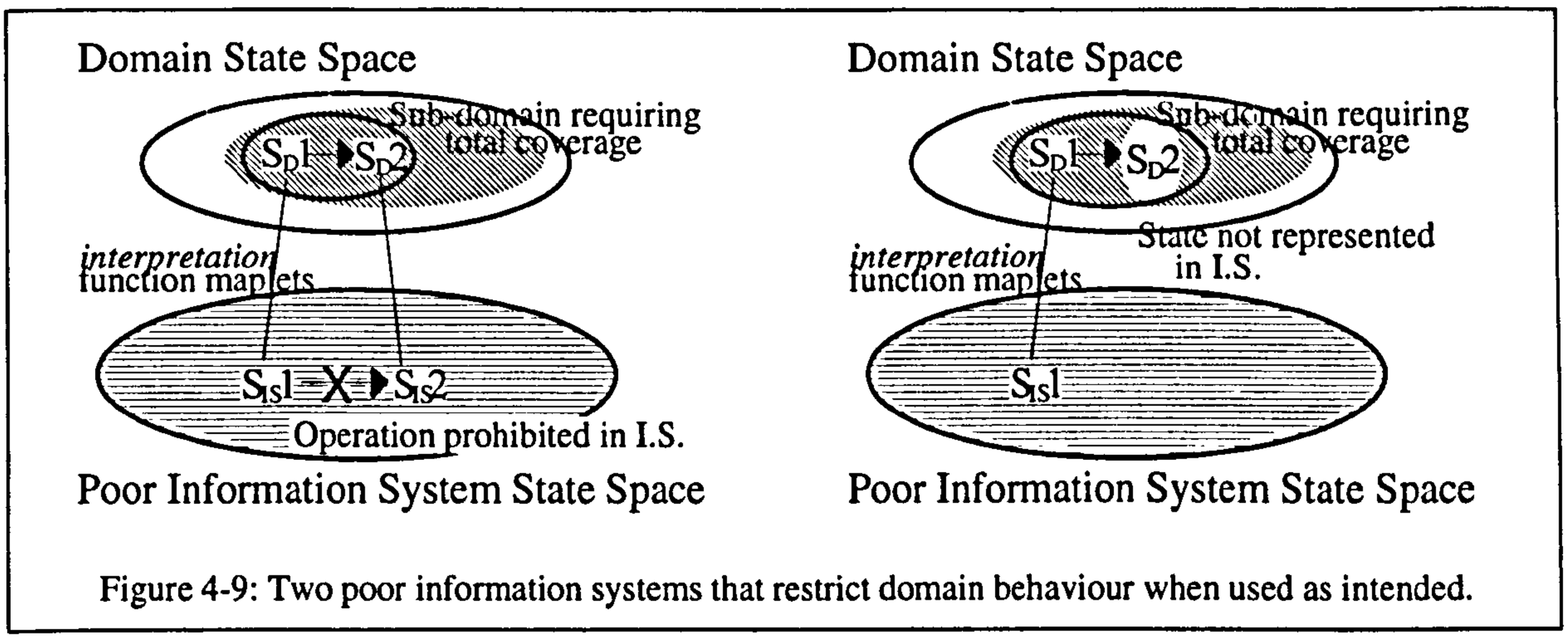


Figure 4-9: Two poor information systems that restrict domain behaviour when used as intended.

In the above figure, two cases where intended use of the information system prohibits behaviours allowable in the domain alone. In the diagram on the left, the domain alone supports a behaviour with the pre-state S_D1 and post-state S_D2. The information system does not support a behaviour with the pre-state $S_{IS}1$ and post-state $S_{IS}2$ where $S_{IS}1$ and $S_{IS}2$ are (assumed here to be the only) representations of the domain states S_D1 and S_D2 respectively. Because S_D1 and S_D2 are in the sub-domain where total support is required, when the information system has been installed, if it is used as intended, the behaviour which moves the values of the domain from S_D1 to S_D2 will be prohibited. The diagram on the right is slightly simpler. In this case, the domain alone supports a behaviour which takes the values of the domain from the state represented by S_D1 to that represented by S_D2. The state S_D2 is outside the scope of coverage of the information system and so again, the behaviour is prohibited if the information system is used as intended.

A good example of the prohibition of certain classes of behaviour when the information system is used as intended can be found in the interaction theory that describes the support that the appointment system provides for the clinic. This is discussed in some detail in Section 12.4.6, but we can summarise the main issues here. The intended use of the appointments system is to provide complete support for certain types of activity. This means that some types of activities can only be booked on the computer system, and that there is no parallel paper based system in these cases. This constraint is specified clearly in the interaction theory. If we inspect the appointments system, we find that there are indeed some behaviours that are not supported by the new system that were previously possible. Notably the computerised system does not support the assigning of estimated durations of appointments at the time of booking, nor the allocation of different types of activity to the same clinic list in a flexible manner. If these functions are important to

the running of the clinic, then the intended use of the information system would be inconvenient and obstructive. Of course, it might be that these functions are not important, or even that the behaviours that they support are never observed in which case we could say that the information system is a better model of the organisation than the domain theory is.

Conclusion

In this section we have investigated two opposing arguments that we should bear in mind when deriving the requirements of an information system from a domain theory. The first argument urges us to create information system specifications that are as simple as possible. This is so that the resulting design is easy to implement, and that the installed system is more usable and more easily maintained than would be the case with a more complex specification. The opposing argument encourages us to create specifications that closely resemble the domain theory. The two arguments or points of view must be weighed against each other when deriving the specification. There are a number of areas where conformity with the domain theory is especially encouraged. These have been grouped together to form four guidelines, or motives: so called as they provide the motivation and consequent justification for the decisions taken during the design process.

The four developmental motives can be understood by comparing them with the requirements of the specification reification process. In particular we can see similarities between the motives and the adequacy obligations of the *retrieve* functions used in reification of abstract specifications. In this analogy the abstract specification is equated with the domain theory, the concrete specification with the IS specification, and the *retrieve* function with the *interpretation* function. The major differences between the developmental motives and the adequacy obligations are caused by their partial application - the motives are just that, not rigid rules and guidelines and the decisions taken in the development process are subjective. The fact that the IS is almost certainly only partly to be a representation of the domain theory means that there are many differences between the reification of abstract specifications and the derivation of those specifications in the first place.

In this section we compared each developmental motive with the appropriate reification obligation, and discussed what it means in the context of the partial representation of the domain theory.

13.4.3 Criticism

Introduction

From the complexity and subtlety of some of the above arguments, we should not be surprised to find that in practice great care must be exercised when deriving information system specifications under the influence of the four developmental motives. The motives are not as formalised as the specification adequacy obligations, and it is sometimes difficult to see whether or not a particular decision 'satisfies' them or not. In particular, some of the motives hinge on comparisons of states and behaviours of the information system and domain combined in a manner described by the interaction theory and separately as this is the only way that flaws in the interaction can be identified and understood. This inevitably adds to the complexity of the process greatly: understanding the behaviours of the domain and the information system together and in isolation is an order of magnitude more difficult than understanding the behaviour of the domain alone. In addition to this complexity we are confronted with a vast array of possible choices concerning the shape and form of the information system. We cannot contemplate all of these choices even superficially, let alone with the detail advocated above. The decision concerning the area of the domain that should be given greater and more sophisticated support is thus extremely subjective.

All these are problems caused by the complexity of the process which become easier as the process itself becomes more familiar. There are more intractable problems however that are less easy to address. These are discussed briefly below.

Difficulty of Specifying IS

One of the features which have made the techniques described here so useful is the support they provide for the composition of existing systems together in such a way as to create a single integrated information system that can be 'greater than the sum of its parts'. This was done in the design of the first version of the DIS (see the specification of DIS1 in Section 12.3) which integrated the clinical record system with the outpatient appointment system. In all probability, almost any system design in an organisation that is already heavily computerised will involve a degree of systems integration with any new automation having to sit with a number of legacy systems, providing information to and retrieving information from their databases.

In order to integrate existing systems using this method, we need to have a good understanding of their behaviour. The legacy systems need to be 'reverse engineered' to form a specification that can then be used as components of the specification of the integrated system. The need for this reverse engineering is clear, but in practice it is a difficult and error prone process. In a sense, the derivation of a specification from the system is similar to the construction of the domain theory - a theory is posited and can be refuted through experimentation with the programme being described. In practice this can be very difficult, especially if access to the system is limited (as will undoubtedly be the case with operational systems) meaning that the best that can be done is to discuss its functions with systems managers and other experts responsible for its running - there is clearly much scope for error if this route is chosen.

In addition to the difficulty of getting a good specification, it might be that one that is strictly accurate is not very illuminating if it does not describe the manner in which the system will be used. Many information systems can be configured after construction so that a generic system can be tailored more closely to the users' needs. In the limit the flexibility provided by this configurability can be very great and can totally change the nature of the system. A case in point is a standard modern word-processor. Most of these come with an advanced language that can be used to create 'macros' that can then act on the text of the document being processed. These macros can completely change the nature of the programme, feasibly changing it from an unstructured text editor to a highly structured information storage system. An accurate specification of such a programme would have to be extremely complex to reflect the extensive configurability catered for. Such a specification would not demonstrate the extreme semantic simplicity of the basic word-processor concept: merely storing and displaying unstructured strings of characters. To a lesser extent, information systems such as those used in the hospital have to be flexible to cope with future developments. While generally not going to the extent of providing a programming language as part of the system, the extra semantic richness created by the structures needed to support the different configurations can still obscure the underlying form.

In the case of the information systems examined (the clinical record system and outpatient appointment system), the full extent of the functionality was not described, and in fact some assertions were made that do not strictly hold. This was so that the system as it is envisaged could be investigated rather than all possible configurations. Thus the data structure of the clinical record system supports multiple inclusion of visits (a visit can feasibly be a part of more than one 'higher' level visit), but in its current form, with its current configuration, the visit graph is actually a tree. Thus the limitations of the system as described in Section 11.2 are not a reflection of the system design, but rather of its envisaged configuration and use.

Although we can see that there are ways round these problems, they reduce the power of the arguments we can rest on the specification of the legacy information systems and the corresponding interaction theory.

Dangers of Relying on Domain Theory

The developmental motives described all rely on a robust and 'accurate' domain theory. If the domain theory is flawed, then so too will the derived information system. In particular, if we are influenced heavily by the motives which encourage us to reduce entropy and behavioural prohibition a poor domain theory will compromise the resulting system design.

In the case of the reduction of entropy, we saw that an information system should not present operations that are forbidden in the domain theory. If the operation is forbidden in the domain theory, but it nonetheless becomes apparent that the corresponding behaviours are actually observed in the organisation, then any information system that does not offer operations deemed illegal by the domain theory will not support this new area of behaviours. In general this will not be a problem, but in extreme cases it might render the information system useless or at least seriously inadequate. To cope with this, most information systems provide functions to the system administrator that are forbidden for the normal user. Simple screens that support the maintenance of single files might be examples of these. This is a pragmatic solution to errors resulting from an excess of hubris during the design of the system, but it is hardly elegant.

Similarly, we might be encouraged to facilitate a greater flexibility in an integrated system than is evident in the constituent components by virtue of the fourth motive - reduction of prohibition. As we have seen, even if the theory is correct, there may well be enhancements that can be made through making it bolder. Indeed, it is unlikely that any abstract theory has represented all the constraints that operate in as complex an environment as a hospital department, and there may be many behaviours that are allowed in our description but are not (and can never be) observed. If an operation is allowed in the domain theory alone but prohibited when the domain and information system are interacting, an assumption of inadequacy of the information system is not the only conclusion we can reach. It might be that the information system represents a better understanding of the domain, and the operations prohibited through the intended use of the system represent behaviours that never are observed in the organisation. If this is the case, then any introduction of flexibility into the integrated system will result in an inferior model of the organisation - one that will be difficult to use, possessing as it does a higher degree of entropy.

Thus in the examples we have seen, the hospital's appointment system might embody a better understanding of the way in which clinic bookings are made than the domain theory (though there is reason to suppose that this is not the case - see the discussion in Section 12.4.6). Certainly there are areas where it is clear that the domain theory allows behaviours that are not seen in the organisation. One of these that was discussed in Section 13.3.2 is the support for two activities to be run for the same patient, one that is only pertinent for women, and the other only for men.

Arbitrary Interpretation

A particularly difficult problem that is associated with the construction of an interaction theory is the choice of interpretation. The adequacy or lack thereof of the information system as judged through inspection of the interaction theory depends on the various interpretation functions being 'good'. Suppose we have a data file is-A in the information system which we claim is interpreted as a concept A in the domain theory. The interaction theory might then have an interpretation function IntA where in general

$IntA$: *is-A* $\leftrightarrow\!\!\!\!\!+$ A.

We might find that there are a number of reasons why the information system is inadequate: IntA might be a partial function meaning that illegal domain states can be represented; IntA might not be surjective meaning that some states in the domain are not supported by the information system; IntA might even not be a function in which case a single domain state could be ambiguous and misleading. All these shortcomings depend on us choosing the range of the interpretation function correctly. Suppose in use, an operator of the system did not interpret is-A as the concept A, but rather as a slightly different one, A'. We might find we could construct an interpretation function IntA' where

$IntA'$: *is-A* $\rightarrow A'$

and

$IntA'^{-1} \in A' \rightarrow$ *is-A*.

That is, this second interpretation function is perfect when judged by the four developmental motives described. What we have discovered is that when used in a certain way, the information system has a number of flaws and shortcomings - we cannot be sure that it will be used in this way, especially if it has not yet been implemented. This is akin to claiming that a particular make of lawnmower is poorly designed as it is awkward to use to bang in nails with. This is almost certainly a valid claim for the majority of lawnmowers, but represents a faulty understanding of its intended purpose rather than any design defect in the machine's construction.

In many cases, the intended interpretation of the data-sets in the information system is fairly easy to divine. For example, the *interpretation* functions IntP, which maps the data-set crs-Pid onto a subset of Patients, and IntV, which maps crs-Visits onto a subset of *Activities*, are not likely to generate much controversy. Other *interpretation* functions are more open to debate. Especially contentious are those interpretations that are derived from the more obvious ones via invariants in the interaction theory.

The assessment here of the adequacy of the information system hinges on this derived interpretation: if we have deduced this incorrectly, we have said nothing useful about the system as it will be used, but merely passed judgement on the consequences of an unlikely style of operation.

In short, through the introduction of an interaction theory, we have created another layer of doubt and subjectivity - we can say a number of things about the information system in relation to its intended environment, but only if we have defined aspects of the nature of that information system correctly.

13.4.4 Synthesis

The synthesis of the arguments for and against the method described for the derivation of information system requirements is similar to that which justifies the construction of a domain theory. The assessment of the adequacy of an information system with respect to its environment is extremely difficult - by providing the tools and guidelines (in the shape of the interaction theory and the various developmental motives) we can help to overcome some of the problems - others are more intractable and need other approaches if they are to be solved, if they can be solved at all. As with the last synthesis section, we can address each of the shortcomings in turn and consider how we should consequently use the methods and tools at our disposal.

The derivation of a specification from an implemented system is undoubtedly extremely difficult - all such 'reverse engineering' is hard to do well. However, what is also not in dispute is the need to have a good understanding of systems that are to be integrated if we are to have any hope of success in that task. Any reverse engineering process constructs a specification from an implementation - the only difference here is that the notation used to express the specification is a mathematically formal one rather than an informal entity - relationship - attribute (ERA) diagram or dataflow chart. The formal reverse engineering process is not significantly more difficult than the informal one, although the latter might be more readily supported by commercial tools (although there are a number of tools that offer a degree of support for the formal reverse engineering of systems such as MALPAS, and the BTool although these tend to be implementation language specific and thus assume that you have access to any source code that might exist).

The problem concerning of which aspects of system to represent in our specification is more difficult and subjective decisions have to be taken. We should bear in mind when choosing which are the relevant datasets and functions to define that it is the intended usage in a particular setting that we are interested in, not some hypothetical system that might result from extensive configuration or other alteration. We can discover this intended or common usage through observation of the system in use if it already is, or discussion with the system designers if it is not. This is how the specification of the clinical record system described in Chapter 11 was derived.

If we are ever going to grasp the role that an information system will play in an organisation, we need a good understanding of that organisation. Correspondingly, if we have a faulty understanding then the value of our cogitations concerning the usefulness of the system will be reduced accordingly. We should, however, be sensitive to the potential for error in the domain theory when making any design decisions, and we should discuss the implications as understood of a certain decision with the would-be users. At this point it might become apparent that the domain theory is in error, or at least insufficiently bold, in which case we should (providing there is time and inclination) return to the theory to correct it. That the theory has been through several cycles of construction and refutation gives us grounds to believe that the number of time errors such as this will occur will be reduced.

We have seen that any discussion concerning the adequacy of the information system as a tool depends utterly on our determining its intended use. This decision is essentially subjective (although it can be guided through observation of users of the system if it exists or with designers if not) and although presented as a problem in the last section, it is a problem that must be shared with all design methods that require an understanding of existing artefacts: if we have misunderstood the purpose of a tool we will not be able to say anything about its usefulness. As is observed elsewhere, by using the formality of mathematics to help formulate our arguments, the decisions made in this regard that might otherwise have been taken implicitly are brought into stark relief: the subjectivity and consequent scope for error must be confronted and addressed explicitly (though of course, explicitly taken decisions might still be poor ones).

An interesting point to make here is that as the domain theory and its constituent concepts have been derived from direct discussion (or 'experiments' - see Section 13.3) with domain participants, we are justified in displaying more trust in these than in those of any information system where implementation considerations will have influenced its form. If the concepts of the information system cannot readily be mapped onto those in the domain theory, this in itself might demonstrate that the entities in the system have been poorly chosen. Before passing judgement however, we should remember that any perceived

reality is capable of being represented in a multitude of ways: although an information system is structured very differently from the domain theory, we cannot automatically assume that it is in error.

Finally, it should be noted that although the method described does not provide us with much help in determining the intended usage of an information system, there are techniques available in other disciplines that address precisely this issue. In particular archaeology and anthropology concern themselves very greatly with the function and usage of tools whose precise purpose has been lost over time. If a society and culture has died out many centuries ago, often all that remains are precisely the 'tools' that supported that culture, where the term 'tools' covers all products that help the socio-economic functioning of the society and individuals (so includes such things as conventional tools, weapons, buildings, jewellery and so on). In order to understand the nature of the defunct society, the purpose of those tools must be deduced. The problem is similar, albeit not so extreme, in living but alien cultures, or even familiar cultures that must be explored in great detail. The techniques that are used to gain this insight can be loosely grouped under the headings ethnography, ethnomethodology, or ethnology. For a brief overview of some of the techniques and philosophy associated with archaeological interpretation, the reader is referred to 'Reading the Past' by Ian Hodder [Hodder86].

In conclusion, we have seen that we can not use the method to lead us inexorably to the perfect system design. Rather we should use it to shed light on areas where discussion with the would-be users and stakeholders of the system might be particularly fruitful. If we use the construction and analysis of the interaction theory in this way, then we have indeed a useful yardstick with which to measure the usefulness of an existing or proposed system (and hence help us design such a system in the first place).

13.5 Issues concerning Construction of IS

Once we have derived a satisfactory specification of an information system, or at least of that component of an integrated information system that is to be constructed, we can set about building the various computer programs that will be the implementation of the specification. The construction of physical, 'concrete', programs from abstract specifications has been an area of study in computer science for many years. Perhaps the most famous early method was that of Dijkstra, expressed in a paper as long ago as 1975 [Dijkstra75]. Since this time, many formal notations and associated refinement techniques have been used. A recent survey [Austin93] showed that the most common of these are perhaps the state based notations VDM and Z, and the temporal languages LOTOS, CSP, and CCS. It is not the purpose of this thesis to contribute technically to the continuing formal methods debate, but for the sake of completeness, some of the benefits and problems associated with their use will be briefly presented. This overview will take the same form as the previous sections: first some of the benefits associated with the use of formal methods will be presented as the justification, then problems that have been observed will be described to form the criticism, and finally the way in which we should approach the use of the particular formal method to derive a working information system is given in the synthesis.

13.5.1 Justification

The technique of formally deriving computer program from their abstract specifications is known as refinement or reification. The process of reifying specifications, and the proof obligations associated with such reification have already been discussed above in Section 13.4.2. The benefits have been widely rehearsed in the literature: it suffices here to present a summary of claims made in support of formal reification in such papers as 'A Justification of Formal Methods for System Specification' [Cohen89] and 'Seven Myths of Formal Methods' [Hall90].

The use of formal methods directly facilitates the construction of a working information system. Through the correct use of the techniques of reification, an implementation of an abstract specification can be accurately and reliably derived.

A derived program can be proven to be a correct implementation, or model, of the specification. The satisfactory discharge of the adequacy obligations of the particular reification method used can greatly increase our faith that a given program performs the functions it was specified to do. This is undoubtedly useful, especially so in the case of so-called 'safety-critical' systems - surely a description of clinical systems.

Software tools can help with both the reification and the discharge of the associated proof obligations. Commonly used formal methods are supported by commercially available tools. These help with aspects of inspection of formal notations and arguments. A few support the construction of and discharge of proof obligations implied by the reification and even the construction of models of the specification as prototypes (for example the B method and tool [Gardiner91]).

The costs associated with the correction of errors is greatly reduced. Because the cost of correcting mistakes in the function of a system increases dramatically with the stage in the system's 'life-cycle', the fact that the implementation can be shown to be correct in terms of the specification, we are able to prevent a class of late occurring (and thus expensive) implementation errors.

13.5.2 Criticism

Not only are the justifications of formal methods well known, but so are the counter arguments (see for example [Goguen90a]). Some of these applied particularly to the project described in this thesis, and it is these that this section will briefly comment on.

Strictly, the reification from an abstract specification all the way to computer code has only been achieved for a few languages that are used for commercial system design (notably Modula). While it is claimed that implementation into the syntax of any computer language can be supported with the help of the appropriate logical framework, the construction of such a framework is in general difficult, and not something to be undertaken lightly (for example, when studying for a PhD). Certain languages map more or less easily onto certain specification notations. The ideas behind the notation used could only be loosely compared with the implementation language chosen for the departmental system: a fourth generation language called Uniface. The major problem lies in slightly different behaviour description philosophies. Whereas the Schuman-Pitt notation relies heavily on the use of invariants to circumscribe the possible states occupied by the system and less heavily on the use of pre- and post-conditions of individual operations, Uniface places more emphasis on the pre- and post- conditions of a number of defined operations, and only weakly supports the use of state space shaping invariants in the guise of declarative entity-relationship rules. Problems associated with the mismatch of underpinning descriptive philosophies are in principle soluble through the expansion of the pre- and post- conditions in the formal notation. This process consists of adding (through the use of the 'AND' conjunction) the invariants of the system to both pre-condition and post-condition of each operation. In practice this would result in very cumbersome and incomprehensible expressions that were difficult to implement correctly.

In the event, no formal reification was used, the specification acting as an informal guide rather than a formal director to the construction of the information system. As a result, many of the benefits of the use of formal methods was lost at this stage. This is in common with the rest of the project however, where the construction and discharge of formal proofs was minimal.

13.5.3 Synthesis

The previous two sections on the derivation of a domain theory and subsequent information system specifications present methods that are relatively novel and were used more or less successfully. This section on the construction of implementations from specifications reiterates tried and tested techniques that were not used. We have not totally invalidated the project however. Although formality adds rigour to an argument, an argument carried out without recourse to predicate calculus of other algebras is not inevitably inadequate or erroneous. The existence of a formal specification helped enormously in the (informal) coding of the computer systems that were influenced by the project. These systems are described in more detail in Section 14.6. The existence of any explicit specification, formal or informal, can help guide development of a complex system. If the requirements are stated in clear English, the system is likely to better fulfil its requirements than if no clear requirements were specified at all. If diagrammatic conventions such as ERA diagrams and data-flow charts are used, then the finished system is likely to be better still. As it is, a notation that is more precise than the first and richer than the second was used to create the specification. As long as it was used correctly and intelligently, this specification was at least as useful as an equivalent English or informal diagrammatic description: the fact that it defines the limits of (valid) system behaviour richly and rigorously means that it is actually better than the less formal equivalents. In fact, it is easy to derive the less formal specifications from the formal, and this to a certain extent was done (at least for the ERA diagrams) for the systems developed in the day centre.

13.6 Issues concerning the use of formalism

This section is not structured in the same way as the previous four. It does not concern one of the stages in the system design method but rather comments about those tools that were used throughout the project: formal methods.

Although there are many benefits to be accrued from the use of formal methods, there is a price that must be paid in terms of intensity of labour. It can be extremely difficult to record as set-theoretic invariants and pre and postconditions notation what is easy to think in your head. While this becomes much easier with practice, the process of formalising informal thoughts will always be hard work. In order that any benefit be gained at all from the use of the formalism, the resultant theory or specification needs to be inspected for inconsistency. Although formal proof was not used, and even semi-formal mathematical arguments were rare, careful inspection of the axioms of the theory to discover inconsistencies or undesirable properties was performed throughout all formal stages of the project. This inspection, and the consequent reworking of the invariants and pre and postconditions was difficult, longwinded and tedious.

Much of the difficulty encountered was precisely caused by the rigour of the notation that we have earlier claimed to be a significant advantage of formal methods. This is indeed the case: when used correctly, the formal notation helps prevent 'floppy thinking' which although useful in everyday life, is the curse of the computer system designer. Unworkable ideas are exposed as such, and must be corrected. The concepts behind the description being worked on, be it the domain theory or an information system specification, need to be sorted out mentally before they can be satisfactorily committed to a formal record. This is hard to do, and can be unpleasant and exhausting. It does mean that the resulting information systems are based on concepts that are more considered than is often the case. Equally, unpleasant though it may be, the identification of an error and the correction of such at this early stage is significantly cheaper than it would be if it happened later on in the system lifecycle.

The use of set theory means that concepts can be represented very precisely, and thoughts that start as vague notions are sharply defined by the time they are committed to paper. The obverse of this argument

is that re-conceptualising an idea from the formal description can be hard. The cryptic and non-intuitive nature of the notation, allied with the rigour with which concepts are recorded means that it is difficult to read in a way that leaves us with a genuine understanding of its semantic content. This is partly because we as humans are a lot happier thinking vague and ill-defined thoughts, and partly because the semantic content of many of the theorems and axioms is so great. It is much easier to understand an ERA diagram than a Schuman - Pitt schema describing the same domain, but then there is much less to understand in the first place.

In short, it must be recognised that the benefits that were enjoyed through the use of a formal notation throughout the project were only enabled through hard work. Harder work, such as the extensive use of formal and semi-formal reasoning, would have delivered greater benefits. It is generally accepted that errors in system design are more usefully and cheaply dealt with if they are caught in the systems analysis phase than if they are only observed during the system construction phase, or even once in use. Any method that is superior by this token will mean that later stages in the software lifecycle are cheaper, faster and easier to complete. It will also mean that the earlier stages are more expensive, slower and more difficult to deliver. The justification for making the job of the systems analyst so much harder is the reduction in cost and shortening of the timespan of the overall system development and implementation project.

13.7 Conclusion

The message from the preceding sections in part four of the thesis is clear. Were the method described foolproof and complete, the resulting computer systems would be brilliantly suited to the job for which they were designed. However, each stage in the process is flawed, and the final system will suffer accordingly. This should not lead us to despair or tempt us to abandon the method. As was argued in Section 4.2, the process of systems analysis, specifically requirements analysis, is very difficult, and yet if it is carried out poorly it can and does lead to appalling faults in the implemented system. In the same section it was claimed that expensive failures are common and persistent: it is accepted that these failures would be prevented through the application of successful requirements analysis. The appropriate interpretation of these observations is not that the myriad system analysts and designers working are going about there jobs incorrectly or negligently, but rather that the task itself is extremely difficult, verging in many areas on the intractable. In this environment, a method that can be used to help improve the analysis conducted, however marginally is a useful contribution. Equally, a method that purports to provide the 'answer' to a problem that continues to baffle the entire computing community should be treated with great scepticism. The author claims that the method used can identify classes of error that might be missed through the use of a less formal or rigorous method. In the previous chapter, we have seen how the method is rigorous and 'scientific' (more so than many conventional methods) and explored which classes of error it is particularly good at trapping. Additionally, by presenting the flaws of the method, we can use it in a more enlightened manner, and can deploy additional methods to cover the areas inadequately or erroneously addressed.

Joseph Goguen talks about the tendency of formal methods practitioners (and indeed the entire scientific community) to deny the existence of error [Goguen90b]. The author believes that insofar as this is the case, it is because the essence of the scientific method has been misunderstood. The way that this has been interpreted in the project presented in the thesis is as a means of discovering errors, and benefiting from them in a structured and rational manner. Error has not been denied - it is seen as an inevitable 'evil', but through its discovery and understanding, we can increase our knowledge of the world and the artefacts we are making to operate within it.

Chapter 14: Conclusion

14.1 Introduction

This chapter concludes the thesis, and in so doing argues that the goals of the work have been met. Firstly the objectives of the project are restated. Each of these objectives is then considered and is shown to have been satisfied. As a result it is argued that the hypothesis of the thesis is correct. The major findings of the project - its salient features - are then spelt out. It is these findings that are considered to be the major contributions to the discipline. These findings are:

- the necessity of three theories - a domain theory, an information system theory, and an interaction theory;
- the benefit provided through the use of (a version of) the scientific method to derive a description of the domain;
- the four developmental motives which balance of the need for simplicity of information system design;
- the domain theory as developed and presented; and
- illumination of a number of philosophical flaws in this and other analysis methods.

Some of the benefits that have already been drawn from the work are then discussed. These are: as an influence on the design of the Diabetes and Endocrine Directorate departmental computer system; as the starting point of an analysis for an out-patient contract management support system; and as exemplary work for the IMC of the NHS Management Executive, used in their search for further development strategies for the Common Basic Specification.

A number of avenues for further work, building on that described in the thesis, are then discussed. These are of both practical theory improving, and methodological in nature. Finally the chapter is rounded off with a conclusion.

14.2 The Objectives Revisited

The original requirement resulted from a long chain of changes in the nature of the health service, and the role of St Thomas' Hospital within it. To be brief, a need was perceived for a computerised information system that would 'support the business of clinical directorates in the hospital', a clinical directorate being the name for an organisational unit roughly corresponding to the earlier hospital departments. The project reported in this thesis represents work towards a design for such an information system. The extreme difficulty in the design of clinical information systems (as evidenced by their high failure rate) and the fact that the project took the form of a PhD meant that there was both the need and the possibility for subjecting the problem to longer and more intense scrutiny than might be usual in such cases.

One of the first observations made that shaped the work of the project was that when it is in use, an information system is interpreted into the world by its users. The symbols and terms displayed to the users are seen as representations of elements and entities in the world as they perceive it. Thus in order to assess the worth of an information system, we must judge how effectively it will be interpreted into the

world: to this end we need a representation of the world, a representation of the information system, and a means of judging one against the other.

Two points can be made which help us successfully to implement such a strategy. Firstly the representations must generally hold (they are universal statements), rather than specifically recording a single case (they are not singular statements). Secondly if the representation of the world, or domain, is to have any credibility it should be derived using the method of empirical science (or a method based on this).

We justify the first point as follows. The designed component of an information system, the 'program', is a universal statement - it is a set of rules which constrain the behaviour of the system in certain ways according to the 'input'. Although we can derive a good singular picture of the world as it might happen to be now (or might have been at one time), this will only be of use in assessing the worth of the information system design at a certain point, rather than generally. If we are to achieve a general assessment of the worth of the information system design, we need to compare a universal statement of the world with a universal statement of the information system - in practice a restatement, or 'projection', of the system's program. In this thesis these universal statements have been called theories. To assess the interpretational adequacy of an information system design we need a theory of the domain, a theory of the system, and a comparison of the two. This comparison which also takes the form of a theory has been called the 'interaction theory' as it attempts to show how the information system will interact with the domain.

The second point follows from the first. A good theory of the domain in which we are interested - namely a clinical unit - is an extremely difficult thing to derive. Information systems are notoriously difficult to get right: it would seem that clinical information systems are even more so. This is partly because the understanding of the user's domain by the computer system designer is inevitably flawed - the analyst's understanding of the domain will be very different from that of the user, and will contain many preconceptions and plain errors. Additionally, in the case of the clinician especially, and perhaps also with other users, the 'universal' language of the analyst (needed to derive a computer system design which is a universal statement) is very difficult to converse in. Clinicians (and perhaps other users) would appear to be much happier talking in anecdotal singular form (this is a simplification - another parallel argument has it that no matter which form of language clinicians use, we can rely on the singular more than the universal). It is thus not only important, but also highly problematic to root out errors caused by a misunderstanding of the domain on the part of the analyst. In order to address these problems, the project exploited the power of the method of empirical science as described by Sir Karl Popper. This has the benefit of being a framework of thought within which a degree of objectivity can be harnessed to assess the worth of a particular theory (*vis à vis* its interpretation). It is also a bridge between singular and universal statements, thus facilitating the communication between analyst and clinician or other potential user. It is not immediately obvious how the scientific method might be used in this instance, so the adoption of an approach that harnessed the method was one of the objectives of the thesis. In fact, the hypothesis is

That the rigorous use of a method where a semantically rich description of an information system is compared with a similarly rich scientifically derived description of the domain to be supported is possible and can prevent interpretational problems in the resulting information systems.

The phrase 'semantically rich' is inserted as (as has been argued previously) the comparison of semantically poor descriptions of domain and information system would lead to a correspondingly poor assessment of the worth of one as a representation of the other.

There are a number of ways of demonstrating the validity or otherwise of any hypothesis. In this case, three subsidiary objectives were defined which between them test the hypothesis. These are

Objective 1: *A method such as that described in the hypothesis was to be developed.*

Objective 2: *The method should be used to derive a 'scientific' description of the clinical directorate.*

Objective 3: *From the resulting description, specifications for components of the Directorate Information System were to be engineered.*

As we shall see, these objectives have been met, and between them they do indeed validate the hypothesis. In the next section we consider how the three objectives were satisfied.

14.3 Satisfaction of the Objectives

14.3.1 Objective 1: A Method Developed

A method was developed which meets a number of criteria set down in the hypothesis and elsewhere in this thesis.

The method supports the scientific derivation of a domain theory. Through the use of 'experimental interviews', singular statements were elicited from clinicians and other users which served to refute a universal theory of the domain. The theory was not only refuted, but was rendered more refutable (that is, more bold) as time passed. Although the approach is different from that proposed by Popper in a number of ways (such as the lack of support for 'controlled' experiments), the use of the singular reliable statement to refute the universal conjectural one, and the search for ever bolder theories is the crux of the scientific process.

The resultant theories are presented in a semantically rich manner. The use of the unambiguous power of set theory and predicate logic (specifically in the form of the Schuman-Pitt formal notation) means that the theories are significantly more precise than they would be if presented in a more conventional notation.

The domain theory can be compared with the information system theory. This is achieved through the use of an 'interaction theory' where the anticipated interpretation of information system state components into domain concepts is recorded. Criticism can then be passed where there is a mismatch between the information system specification and the domain theory, and remedial action taken (if deemed appropriate).

The method supports the engineering of information system specifications. Not only is it possible to compare existing information systems against the domain theory, but it is possible to posit hypothetical systems, and by using an appropriate interaction theory and the four developmental motives described in previous chapters, move incrementally towards an improved system. The positing of hypothetical structures, the use of defined ('scientific') procedures to assess their goodness, and their subsequent refining is the essence of engineering.

14.3.2 Objective 2: A Description Derived

A theory of the domain was developed over a period of some eighteen months. The theory, spanning over forty pages of set theoretic notation, was restated in its entirety over twenty times. Many of the theorems embedded within it were abandoned after refutative evidence came to light. A dozen or so of these refutations are recorded in this thesis as the original theorem, the refutative evidence, and the reworked theory. After the eighteen months we cannot say that the theory is perfect, or that there is no room for improvement, merely that it is better than it was. The use of set theory within the framework of the Schuman-Pitt notation lends the presentation of the theory both clarity and precision. The way in which the evolution of the theory has been recorded in the thesis means that the reader can see not only what theorems the theory contains, but also in many cases, why it contains them and not certain others.

14.3.3 Objective 3: A Specification Engineered

The specifications of two information systems were considered in the light of the domain theory: that of the Clinical Record System (CRS) and the first fragment of the Directorate Information System (DIS1).

In the case of the CRS the system already existed (albeit not in an implemented form), the interaction theory merely serving to illuminate areas where discrepancies between the system and the domain theory lie (and hence show where there are opportunities for 'improvement').

Although DIS1 did not and currently does not exist, it is designed to be the integration of two components that do - the CRS and the outpatient appointment system (OPAS). The specification of DIS1 is thus a composition of the specifications of the CRS and the OPAS. Few extra entities are introduced in DIS1 that do not exist in either the CRS or the OPAS, the integration largely being concerned with how the state variables of one of the component systems constrain values of those of the other system. The integrated DIS1 was developed with the aid of four developmental motives. These motives influenced the response to inadequacies in the (putative) interpretation of DIS1, thus motivating (some of) the design decisions taken. Again, we do not know that DIS1 is a good information system - merely that it is better, at least in an interpretational sense, than the other possibilities discussed in Chapter 12.

14.4 Validation of the Hypothesis

Having demonstrated that the three subsidiary objectives have been met, we can argue that the hypothesis has been validated with some conviction.

The hypothesis states that it is possible to show that a semantically rich description of an information system can be compared with a scientifically derived description of the domain of activity which the system is to support. Through the construction of a scientifically derived domain theory (it was argued above that for all its differences with classical science, the essential features of the approach are preserved), and its composition with the information system specification by way of the interaction theory we have shown that such a comparison is indeed possible. The use of the Schuman-Pitt notation ensures that the descriptions being compared are semantically rich.

Furthermore the hypothesis states that this procedure is of some benefit in that it can help interpretational problems in the information system. We have demonstrated that this is indeed the case through the recording of theorems from early versions of the theory and their refutation, and through presenting potential interpretational inadequacies (with respect to the domain theory) in a designed information system - DIS1.

What we have not said is that the method used is better than any other at preventing such problems - to argue thus would entail a detailed comparison with other similar systems designed using these other approaches. Even were that possible it would be infeasible (or more likely impossible) to keep all other factors constant meaning that any such empirical comparison would be flawed. Of course, the author would argue that the method described does present the user with benefits not apparent in other approaches. Indeed were this not to be the case the hypothesis would be validated but the work still useless. Other methods do not articulate the problem clearly, and specifically do not compare the information system against the organisation it is intended to support. They do not use a 'scientific' method to understand the organisation, relying instead on discussions with users that have a universal quality, and are thus difficult to understand let alone disagree with. They do not link the information system with the organisation in an explicit manner meaning that we do not know where design compromises have been made and why. Having said all this, it is clear that most of the commercial analysis techniques are significantly easier to use than that described in the thesis.

14.5 Salient Features of the Project

We can summarise the work described in the body of the thesis in another way, by listing the major achievements or findings resulting from, or emphasised by the work. These are briefly presented in this section.

The first notable claim is that in order to judge the interpretational adequacy of an information system, we need three theories. To see how the information system will be interpreted into the world we need:

- a theory of the (user's perception of the) world which is the domain theory;
- a theory of the behaviour of the information system which is the system specification; and
- a theory of how a model of the second theory will be interpreted as a model of the first - this is the interaction theory.

It is only by inspecting the third composite theory that we can reliably find and comment on the interpretational inadequacies of the information system with respect to the target domain.

The second claim concerns the derivation of these theories. The scientific method can to all intents and purposes be used to refine the domain theory. This provides us with a reliable tool with which to judge and improve a theory. It is particularly suitable, useful, and necessary in the creation of the domain theory inasmuch as it:

- enables the preconceptions of the analyst to be tested and where erroneous exposed;
- facilitates the communication between the universal language of theories and the singular language of everyday experience (especially for clinicians); and
- provides a measuring device by which two competing theories can be judged.

The scientific method is not really appropriate for the creation of the information system specification - this is either a projection of existing theory (the system's program) or the statement of the behaviour of a putative but currently non-existent system. Similarly the scientific method is of no great help in the construction of the interaction theory: the only guidance here is common sense and to follow the advice given in Section 11.3.2.

A third methodological as opposed to practical claim concerns the engineering of an information system using the three theories. Here it was argued that balancing the desire for simplicity in the design of the information system was a need for accurate domain representation. In particular four developmental motives, similar in many ways to the adequacy obligations for data reification, have been stated and described which guide and motivate the movement towards domain conformity.

On a practical note, the method has resulted in the creation of an extensive, robust, and sophisticated description of the operational process of the provision of healthcare. Although it is by no means perfect, we know that it is an improvement over other possible descriptions that were refuted during its development and, because of the presentation of the refutations, we can see why it is a better description. It is presented as a sequence of enrichments and refinements of an initial, highly abstract description which represents the process of care as consisting of basic or composite clinical activities and the relations between them. The existence of this theory, or at least the refutations, might well be of use to workers endeavouring to derive similar articles (that is, other requirements analysts, systems analysts, and designers).

One final claim relates to the philosophical points raised in Section 13.3 - Issues concerning Construction of Domain Theory. Many of these are common to all forms of information systems and analysis techniques. The precise articulation of the problem and the use of a formal notation to support the method exposes these 'metaphysical' concerns more starkly. Although it might not at first seem that this is a benefit, the understanding of the limitations of a tool can only increase that tool's efficacy. The illumination of the problems of analysis in general, and this form of analysis in particular means that the method can be used more powerfully (as indeed can other methods).

14.6 Preliminary Benefits

Although the major specification produced as a result of this project, namely DIS1, has not been implemented, the work conducted has nevertheless been beneficial in a number of areas. This section describes these.

14.6.1 CRS design influence

As has been previously explained, the department's clinical record system (CRS) was being re-written at the time of the project. Specifically the data modelling phase of the system's re-design was conducted during the latter stages of the domain theory development process. Although the 'correct' time to consider the data architecture of a system is (according to the method being espoused here) after a tentative first pass specification and related interaction theory, the domain theory alone can help us in an informal way with system design.

In this case it is claimed that the domain theory influenced the design of the data architecture of the CRS. Although the earlier version of the CRS - called 'APL-Diabeta' - that was being updated placed the clinical visit at the heart of that system's data architecture, its central role was reinforced in the new design. Almost all the attributes of patient health and activity are now directly and primarily associated with a visit record. This is in many ways similar to the structure of the domain theory where the activity plays such a significant role.

Another area of similarity with the domain theory that was not present in APL-Diabeta concerns the existence of an 'inclusion' relationship in both the new CRS and the domain theory. The APL-Diabeta system and early versions of the new CRS data structure recorded the clinical type of a visit as an attribute

of the visit, but that was the only 'structural' information pertaining to the visit supported. At display time a filter could select whether details of visits of a certain type or all types were displayed. The problem came with the pregnancy episode. During the period of pregnancy the patient will see a doctor and nurse much more frequently than they would normally meaning that 'recent' visits would have a different meaning if the patient was pregnant compared with the normal state of affairs. This was considered undesirable and therefore a problem. Creating new visit types for all visits during the period of pregnancy was one solution, but this would have been a crude one as much of the activity conducted during the pregnancy is in fact the same as a visit to that specialist outside pregnancy only at a higher frequency[xxvii]. The approach taken is similar to that used in the domain theory which is to have hierarchies of visits such that one can be included in the other. This has effectively given an additional, and more flexible, method of categorising and aggregating visit records.

Although these changes might well have been suggested and implemented without the use of the theory, it existence meant that the identification of these 'solutions' was more inevitable than it might have been. At the stage of data architecture design, any contributions to the debate must be welcomed as useful - mistakes made at the beginning are difficult and expensive (in terms of time at least) to fix.

14.6.2 The Out Patient Contract Management Support System

One of the areas of direct concern to the Diabetes and Endocrine directorate was the introduction of contracts for services provided. In patient activity has been the subject of contractual agreements with purchasers since the introduction of the NHS and Community Care act in 1990. There are a number of different forms of contract, including Block, GP Fundholder, and the so-called 'Extra contractual' which is really a contract that is negotiated on an *ad hoc* basis and not as implying a standing relationship between purchaser and provider. It is vital that patient activity is tied to contractual income if the service that the hospital delivers is to be managed rather than just administered.

The clerical burden associated with the setting up, monitoring and discharging contracts is small when compared with the cost of a typical in patient episode (which may cost thousands of pounds) but large when compared with the cost of an outpatient consultation (which may cost under one hundred pounds). For this reason out patient activity has generally been designated as an overhead to be distributed across all in patient activity. This is a reasonable model for the majority of directorates where out patient activity is a necessary precursor and follow up to in patient treatment. For the Diabetes and Endocrine Directorate where the majority of activity is out patient based, this is an unwise not to say dangerous approach to take. The whole directorate would become an overhead on the rest of the hospital, and its closure would result, on paper at least, in significant efficiency enhancement for the other directorates. For this reason a contract management and support system was needed in the directorate, and because of the low cost of the services being contracted it was important that the procedure was as automated as possible.

This is the background to the stated need for an Out Patient Contract Management System (OPCMS). The system itself was not specified in Schuman-Pitt notation, but a theory of the organisation as it would be once contracts were efficiently supported was. This is an example of a hypothetical domain theory - a theory of an organisation that does not currently exist, but rather the description of the behaviour of an organisation that it is imagined could exist. In this case it is the theory of a mainly outpatient clinical directorate which has an efficient contract management system in place. The hypothetical domain theory

[xxvii] The approach used by APL-Diabeta was even cruder: the pregnancy module of the system was essentially an entirely distinct 'clone' of the conventional system, specialised for pregnancy care, and with no communication with the conventional system.

was created as a refinement of the domain theory (in fact a refinement of a specific class in the domain theory) and is presented in Appendix 6.

The theory of the organisation with an OPCMS in place was used by Sanjay Sanghrajka, a BSc student from the University of Surrey, as the basis for a final year project he conducted for the department [Sangh94]. This analysed and documented the requirements for an OPCMS. Following the successful completion of this project the author designed and constructed a prototype OPCMS as an extension of the new CRS system.

14.6.3 IMC interest

The Information Management Centre of the NHS has for many years concerned itself with (amongst other things) the definition of data standards for use by the health service as a whole. One of the data standards is called the Common Basic Specification (CBS) [IMC92], discussed earlier in section 7.2. The development of the CBS has developed over the course of four years at a cost of as many millions of pounds. The work was suspended following the publication of a report, written by the management consulants CASPE, which questioned the usefulness of the work and the way in which it was evolving. On publication of the report, further development work was stopped and a number of projects established to determine the effectiveness of and future development strategy for the CBS. These are known as the CBS demonstrator projects - the outcome of the projects is being reviewed by the CBS Assessment Board [IMC91]. The projects have revealed a number of shortcomings. One of these is that the wealth of knowledge and experience built up as a result of so many man-years of analysis work does not reveal itself through the data model as presented. This is because the semantics of the presentational medium used have been too poor [Cohen93]. It is possible that the use of more formal notations would address this particular problem.

For this reason the IMC has expressed a degree of interest in the work reported here, attempting as it does to present a formal theory of a small part of the health service. It is interesting not just as an exemplar of the sort of work that is possible, but also because some of the primitive concepts in the CBS also appear in the domain theory described in this thesis. Notable similarities is the decision to represent medicine as a hierarchy of nested activities (although those of the CBS are not exclusively clinical), and the association of all activities with a subject (although those of the CBS are not exclusively patients). It seems that the CBS is richer than the domain theory (and certainly more expansive) but less rigorous.

At the time of publication of the thesis, the future of the CBS is uncertain. However, a PhD project has been set up with the involvement and blessing of the IMC[xxviii]. This is looking at the value that might be gained from formalising the CBS, and the difficulties that will be encountered in trying so to do. This work has taken heed of, and to a certain extent benefited from that reported in this thesis.

14.7 Further Work

As has been alluded to several times in the thesis, there are a number of ways in which the work described might be continued and expanded on. Both the domain theory itself and the method for deriving the information system specification would benefit from further work - this section comments on a number of areas that might be of particular interest.

[xxviii] This work is being conducted by Max Jones at the Department of Life Sciences at the University of Nottingham

14.7.1 Towards a More Generic Domain Theory

Although much effort was made to ensure that the theory was general and applied to more than one directorate, the claim that this effort has been successful is unsubstantiated. To have any confidence that the theory is indeed general significant time and effort needs to be spent in other departments and directorates. The results of analyses in other directorates would be specialisations (such as that included in Appendix 3) that describe the particular conditions and services in these organisations. It might be that the derivation of such specialisations is possible - it is more likely that some change to the structure of the theory is needed before we can be confident that it is indeed generic.

One area where both specificity and genericity might be combined fruitfully is that of abstract activity classification. In the theory as it stands the set *Activities* is partitioned into *In & Out* and *Request, Proceed, & Complete*. However, the theory makes special provision for other, derived subsets of Activities, in a number of places. One of the derived subsets is that sort of activity that can be considered to be 'concrete' rather than abstract, and occupies the time of a member of staff of the organisation. Thus Init Dr Consultation is a more concrete activity type than Diabetes Care, and the former needs continuous resource to run where the latter does not. The addition of a new partition of Activities, say Abstract and Concrete, and the development of properties of these subsets might be a valid and useful way of representing domains as percieved by their workers. The development of this concept would be an emboldening and would so lead to greater specificity. However, it is imagined that this is an example of a property that is shared by all clinical domains and so is an example of (proposed) genericity. Representation of the concept of concreteness might greatly improve the theory's descriptive power, and whether valid or not the investigation of the area would surely lead to valuable insight.

14.7.2 Towards a More Elegant Theory

There are a number of ways in which the theory could be improved so that it describes much the same system but in a neater and clearer way.

Firstly the theory could be semantically identical but syntactically neater. An example where this might be useful is the definition of the structure and behaviour of the state component EmbedType. This is currently declared as

T 16: $EmbedType: \mathsf{Pr} \pfun (TGroupers \pfun (Types \rightleftharpoons Types))$,

an ugly and difficult to grasp quartet which in turn renders the expression of the invariants that constrain it bulky and complex. A little thought is needed to turn this and other over complex structures into more simple concepts. Another area where some re-expression might be useful is in the definition of time in the Clock class of the theory. Here the use of sequences and their associated operations might make the introduction of time simpler - as it is a lot of effort has gone into defining and needs to go into reading a part of the theory that does not say anything interesting about medicine. Just as not all the power of set theory as developed in the literature has been exploited, nor has all the power of the representational medium. The theory as presented makes little use of the 'object-oriented' properties of the Schuman-Pitt notation. Although the properties of a class in the domain theory are inherited by its subsidiaries, none of the other properties of the object oriented paradigm is used. There are no 'objects', no data and process encapsulation, and no methods passing between processes. The recasting of the theory into a more object oriented form might yield valuable benefits: it has been widely claimed that object oriented programs and specifications are significantly easier to understand (although whether this is on an intuitive or formal

level is not clear) and develop. Any alteration that lends conceptual simplicity to the theory is to be greatly welcomed.

Secondly there are a number of areas where the semantics could be changed slightly so that the theory was sleeker and clearer. Currently the fact that the suspend operation returns an activity to the set *Request* causes problems both in the theory and when it is composed with information system specifications. In the theories a suspended activity is often treated differently where it can be (a request represented by a visit record must have been suspended for example). This could be formalised by setting up a new subset of activities - Suspend - which was similar in many ways to *Request*, but could be treated differently where appropriate. The fact that the division into three disjoint subsets in the first class of the theory means that the reworking required to effect this change would be significant. Another problem is that concerning concurrent activities run by the same clinician. Although the rule originally suggested in the theory was refuted, it was not replaced by a different one and the constraints are thus very weak in this area. It is clear that some form of limiting theorem should be introduced (a nurse will not run a patient education session for one patient while at the same time be prescribing insulin to another for example), but not what form it should take.

14.7.3 Changing the Rules During the Game

The theory as it stands allows for one specialisation for any of its models. There are many specialisations that are forbidden by the invariants in the specialisation classes, but once one that is permitted is chosen there is no facility to allow for it to evolve. This is clearly unrealistic as the organisation that the model purports to describe will undoubtedly change its nature over time. For example, new types of clinical activity might be offered, new types of blood tests might be conducted, representatives of new professions might be employed by the organisation, and the professional structure of the organisation might be changed.

The issue of 'changing the rules as the game is being played' is an extremely difficult one to address. The slow speed of change of the organisation when compared to operational behaviour means that further experimental interviews with clinicians might not be as effective (as noted earlier in Section 13.3.2). One way of overcoming the problem might be to suggest a number of pragmatic alterations to the model and see what forms of change these prohibit. The prohibitions could then be discussed with clinicians to see if any were likely to be observed. The theory that resulted would probably not be as 'realistic' as the operational behaviour described by the theory, but at least it would be one that worked.

An example of a pragmatic alteration would be to allow multiple specialisations, each of which has a start time. The relations between operational and specialisation state components would also specify which specialisation was being referred to. When a new specialisation became valid new members of the operational / specialisation relation would comply with the rules as defined by the new specialisation. The complexity would lie in dealing with activity structures that spanned the change in specialisation. Although a solution could undoubtedly be constructed, that it could represent the domain with acceptable accuracy is less certain.

One thing that should be noted is that it is probable that any change to the theory to accommodate and describe organisational change is liable to be extremely complex and increase greatly the difficulty in comprehending the theory. The law of diminishing returns operates here and we should be clear that a description of the nature of organisational change is sufficiently important to us to justify the extra work required. Bearing in mind this caveat it is nevertheless clear that the theory could benefit from the definition of a framework within which the specialisation state components could evolve, for

completeness sake if nothing else. Such evolutionary invariants might prove significant if the information system to be developed was a management information system which generated hypothetical future scenarios for comparison with the current situation.

14.7.4 Other developments

There are many further ways in which the theory might be improved to give a richer and more accurate description of the organisation. Any development which acts to constrain the possible states of models of the theory (or rather increase the ratio of forbidden states to permitted states) is encouraged by the directives of the scientific method. There are two areas where such development would be particularly beneficial to the description of health care.

Firstly the representation of clinical records could be rendered more sophisticated. In the theory as it stands the description of the health status of the patient is about as crude as it could be - in fact a conscious decision was taken to avoid all aspects of the patient's condition even down to his or her gender. The framework that already exists would be a good place from which 'o launch an exploration of this most central aspect of health care. It has been argued earlier that the difficulties associated with a representation of a clinician's perception of the state of health of the patient will be very hard to overcome. To derive a theory that can account for the state of health of one patient as perceived by one clinician would be an extremely difficult undertaking - to produce a theory that can accommodate all patients and all clinicians would be a task many orders of magnitude less tractable. The problems of 'shared reality' and the subjective nature of knowledge cause even more of a problem in this area.

We should be sure that we know what we are attempting to describe with the theory before we embark on the task of constructing it. We want an abstract set of state components and rules which can be used to describe the state of health of a patient, and which can be agreed on by clinicians. The job of reaching consensus in these matters is that which is undertaken by the whole machinery of professional medicine - teaching hospitals, universities, royal colleges, pharmaceutical companies, academic journals, regulatory bodies, and many other organisations. To suppose that the diverse opinions and understandings that are reflected by this enormous spread of human endeavour can be succinctly encapsulated in a single scientific theory demonstrates not only outrageous *hubris* but also a misunderstanding of the fluidity and ephemerality of any knowledge and consensus that does exist.

Having said this, and recognising that a 'good' theory of the medical aspects of health care is even more unattainable than one for the operational concerns explored in the thesis, the scientific method described here can nevertheless provide useful insight to help in the development of medical computing. While it has just been argued that a generic clinical record system is more or less impossible, such specifications are nevertheless being proposed, produced, and incorporated into information systems used by the medical profession. The testing and refutation of these theories will reveal where their shortcomings lie, help to improve them, and thus help to improve the information systems that have been based upon them. This is a useful pragmatic approach that could be adopted in the development of the domain theory so that it can represent the health of the patient.

A second area where the theory would benefit from more work is in the clarification of the interface between organisations. Although this has been somewhat addressed through the use of the '*In*' and '*Out*' partitions of *Activities*, this part of the theory is nevertheless fairly crude. As the health 'market' continues to evolve in this country, the sophistication of the boundaries between organisations will increase, as will their importance to medical organisations, large and small. The historical, and relatively simple, relationship between secondary, primary, and self care is likely to change radically and be replaced by a

much more involved, complex and fluid system. Information systems will increasingly need to recognise and adapt to these changing relationships - an abstract theoretical understanding of them could help enormously. This then would be a fruitful area in which to invest time and effort so as to gain insight and to enable the robust design of the necessary information systems.

14.7.5 Further Investigation into Hypothetical or Imaginary Domains

One area that has not been discussed much in the thesis is the use of information systems to facilitate and encourage organisational change. One of the assumptions of the method is that the information system is a fairly passive thing as far as the effects on the organisation to be supported are concerned. The behaviour of that organisation, it is supposed, will not fundamentally change once an information system has been introduced. We have taken the rules and state components expressed in the theory from the behaviour of the domain as it is perceived by clinicians, and seen to what extent a proposed information system might support or inhibit this. We have not imagined that the introduction of the information system will enable and directly cause the introduction of entirely new state components, and change the rules between those that already exist.

It has been shown, however, that the introduction of information systems, especially those of an operational nature, can radically effect organisations and the way in which the workers in those organisations perceive them [Scott91]. As a result of the introduction of systems, the balance of power can be dramatically altered, flows of information can be enabled or inhibited, and new roles and functions of the organisation can be revealed. Some analyses go so far as to compare an organisation to an information processing device [Morgan86]. If we accept this, we should not be surprised if the introduction of an automated information processor jolts the organisation with sometimes unpredictable effects.

There are number of ways in which we might explore the changes that information systems can wreak on organisations. We can posit a new domain different from the current one, and design an information system to support this in the hope that it will then influence the organisation to adopt the forms described in this imagined domain. This is what has been done in Appendix 6. The new domain is essentially the same as the current one with the addition of a few new state components to reflect the introduction of outpatient contracts to the directorate. Although the directorate currently attempts to support the contracting process it does this inefficiently and inflexibly. It is hoped that the implementation of an information system that is a representation of the hypothetical domain described in Appendix 6 will change the organisation in such a way that it accommodates and indeed exploits the 'opportunities' presented by the introduction of health care contracts.

Another approach is to use information systems to introduce much more sweeping organisational change: this is one of the goals of the technique known as Business Process Re-engineering (BPR) [Hammer93]. Although this is not the subject of this thesis, it is recognised that it is an area of extremely lively academic debate. The use of formal methods can help significantly in providing an insight into the behaviour of the re-engineered organisation. Interested readers are referred to the work of Holden and Glykas (see eg [Holden93], and [Glykas94]) for further illumination in this fascinating subject.

The incorporation of hypothetical domains into the approach described in this thesis would much increase its flexibility and power. There is similarly no reason to suppose that the continuing debate on the subject of BPR would not benefit from some of the findings of this thesis. It is certainly an area where further work might usefully be done.

14.7.6 Developmental motives and Information System Representation

The method as described exists in varying states of development. The use of the scientific method to derive and refine the domain theory is fairly well developed, and is presented as a (reasonably) mature and central part of the thesis. This is not surprising as by far the majority of time was spent on this part of the project, and the problems associated with it were scrutinised most closely.

Equally important is the derivation of the interaction theory and the consequent engineering of the information system specification. These areas have been discussed in the text, the developmental motives in particular at quite some length. However, it is clear that there is much work that can be done here to elucidate and clarify these topics. Specifically the way in which we should construct the interaction theory is very under-developed: although the germ of the desired approach is presented in Section 11.3 there is much more substance to this issue than is discussed and this can be and should be examined further. In a similarly manner, the developmental motives merit more exploration. In particular the precise way in which they differ from the obligations of reification as a result of pragmatic concerns needs to be more authoritatively catalogued. As part of this process, the rationale for the use of each motive, and guidance for when each should be used should be reinforced.

Further problems exist with the representation of the information system. In the work described here various subjective judgements were taken in constructing and recording the specification of the clinical record system and out-patient appointment system so that it was possible to construct an interaction theory. In particular, the distinction between operational and specialisation state components was made for the information systems such that the specification could be linked to the domain theory to form the interaction theory. The problem with this was that some of the general rules of the domain theory were actually expressed as specialisation states in the information system (at least this was the case with the clinical record system). In other words the rules were not static and could be changed, or re-configured, during operation of the system. In short, the information system is more adaptable than domain (as perceived and described). Guidelines for the reverse engineering of such adaptable information systems (and some languages lend themselves to much more adaptable implementations than the fourth generation language used) should be discerned, catalogued, and presented.

Another area worthy of closer regard is that associated with delays in the state of the information system. The interaction theory presented records an intended use for the information system where changes in the state of the system reflect simultaneous changes on the state of the organisation. Very often an information system is not used in this way: rather the database is interrogated in a 'live' manner, with changes to the database input in periodic batches, possibly by secretaries or data entry clerks at the end of the day. The effect of such 'delays' between the state of the organisation and that of the information system could be explored using an interaction theory, but such a theory would be more complex than that presented. In particular the invariants covering intended use would have to be cast with great care. Such an interaction theory would help to reveal the implications of the use of old data on the behaviour of the organisation, and illuminate problems and issues that would have to be addressed in the design of the information system.

14.7.7 Possible development as general service model

Finally it might be worthwhile to explore how the domain theory could be developed so that it described non-medical organisations. Many of the early aspects of the theory are not specific to medicine but could equally well hold for any organisation concerned with providing services of differing types. In this sense

the early class schemas could be used as tool for exploring and understanding a number of such service oriented organisations.

We should not assume that such generality exists, or even that it would lead to useful insights. However, if fundamental similarities could be found between service providing bodies the job of the systems analyst would be greatly eased and operational information systems, being smaller variations on a common template, much cheaper to procure. Any project to investigate development in this area would have to be very shallow, or very large, as the number of different organisations each of which claims to have totally unique problems and world-views is vast. Nevertheless such a project might well prove to be rewarding and interesting.

14.8 Conclusion

We are now in a position to conclude this final chapter and with it the thesis. We saw above that each of the objectives of the project have been satisfied. Firstly, a method for constructing an information system specification was devised and used which satisfies the criteria laid down in the hypothesis and fleshed out in Section 5.3. Secondly, the method was used to derive a formally presented theory of the domain of interest - a generic clinical department - in the scientific manner as described in Section 6.3. Thirdly, components of an information system - in this case a Directorate Information system - were specified and formally compared with the domain theory to reveal shortcomings and thus enable improvements to be engineered. It is further argued above that the satisfaction of these objectives means that the thesis' hypothesis has been shown to be correct.

We then saw that the work reported has already been of some benefit in a number of areas, both at the departmental level (influencing the design of the Diabetes and Endocrine Directorate's Clinical Record System and acting as a starting point for analysis of the out-patient contracting process) and to a lesser degree at the national level (being of some interest to the IMC in their running of the CBS demonstrator projects and assessment board). Finally we considered how future work developing the findings of the thesis might be fruitful. This future work would be of both a direct theory enhancing and methodological technique refining nature.

The text of the work is long and involved and many points have been made along the way. However, the most important issues can be briefly re-iterated here. Firstly three descriptions or theories are needed to understand the merits and shortcomings of an information system. These are the domain theory, the information system theory or specification, and the interaction theory. Secondly, the determination of a valid domain theory, although an extremely difficult and imprecise task, is helped enormously through the use of the scientific method (or a modification thereof as described in the thesis). Thirdly, once the information system specification and the domain theory have been composed together to form an interaction theory (according to the technique described in Section 11.3), four developmental motives can be used to 'improve' the specification.

Perhaps more important than any of this is the means of presentation of the thesis. This has taken the form of an extended case study. The findings of the work (of which the most significant have been summarised here) have not been presented as 'givens', but justified through the use of real examples. In this sense it is hoped that the thesis is a revelatory and didactic rather than an instructional pedagogic work: any value and lessons can be drawn out from the text by the reader, albeit that the process is guided by the argument in the text. This applies not just to the methodological lessons learned but also the domain theory itself. By sharing the path to the results with the author, the reader can not only see what the details of the analysis are, but also why they are so.

This work should not be seen as a finished article, but rather a contribution to the debate concerning the design of information systems, particularly, but not only, those destined for use in the medical sector. This debate is an important one: information systems surround us and are influencing our lives to an in ever increasing extent - it is vital that these ubiquitous creations help rather than hinder human endeavour. It is hoped that this thesis helps progress the discussion and proves to be of interest to practitioners in the field.

References

Abel92 Abel-Smith B (1992) *The Reform of the National Health Service Quality Assurance* in *Health Care*, Vol 4, No 4 pp 263 - 272

Alex71 Alexander C (1971) *Notes on the Synthesis of Form* Cambridge, Massachusetts: Harvard University Press

August91 August J. (1991) *Joint Application Design: The Group Session Approach to System Design* Englewood Cliffs, NJ: Prentice Hall

Austin93 Austin, S., & Parkin, G. (1993) *Formal Methods: A Survey*. National Physical Laboratory, Teddington

Austin62 Austin J. (1962) How to Do Things with Words Cambridge, Masschusetts: Harvard University Press

Avison88 Avison D. & Fitzgerald G. (1988) *Information System Development - Methodologies, Techniques, and Tools* Oxford: Blackwell

Avison90 Avison D., and Wood-Harper T. (1990) *Multiview: an Exploration of Information Systems Development* Oxford: Blackwell

Beer81 Beer S. (1981) *The Brain of the Firm* Chichester: Wiley

Bert68 Bertalanffy, L von. (1968) *General System Theory*, Braziller,

Bick92 Bickerton M. (1992) *A Practitoner's Handbook of Requirements Engineering Methods* Oxford: Oxford University Computing Laboratory

Boehm88 Boehm B. *A Spiral Model for Software Development and Enhancement* IEEE Computer 1988 21(5) pp 61-72

Brooks82 Brooks, F. (1982) *The Mythical Man-Month: Essays on Software Engineering* Reading, Massachusetts: Addison Wesley

Brooks87 Brooks, F. (1987) *No Silver Bullet: Essence and Accidents of Software Engineering* IEEE Computer April 1987 pp 10-19

Bullas89 Bullas S. (1989) *Case-Mix Management System Core Specification* London: HMSO

Buxton89 Buxton M, Packwood T, Keen J (1989) *Resource Management: Process and Progress. Monitoring the Six Acute Hospital Pilot Sites*. Uxbridge: Health Economics Research Group, Brunel University.

Carson93 Carson, E., Cramp, D. *The Role of Health Care Modelling in the Development of Knowledge Based Systems for Chronic Disease Management* in Proceedings of the 1993 IEEE EMBS Conference San Diego (1993) IEEE Press

Chamb89 *Chambers English Dictionary* (1989) Cambridge: Chambers Cambridge

Chambers88 *Chambers English Dictionary* (1988) Cambridge: Chambers Cambridge

Check72 Checkland P. (1972) *Towards a Systems-Based Methodology for Real World Problem Solving* Journal of System Engineering 3 (2) pp 87-116

Check90 Checkland P., and Scholes J. (1990) *Soft Systems Methodology in Action* Chichester: John Wiley & Sons

Chen76 Chen P. (1976) *The Entity Relationship Model - Toward a Unified View of Data* ACM Transactions on Database Systems March 1976 pp 457-475

Coad91 Coad P, Yourdon E (1991) *Object Oriented Analysis* Englewood Cliffs, NJ: Prentice Hall

Codd70 Codd E. (1970) *A Relational Model of Data for Large Shared Data Banks* Communications of the ACM 13 pp 5-15

Cohen84 Cohen B. (1984) *The Specification of Complex Systems* Reading, Massachusetts: Addison Wesley

Cohen89 Cohen B.(1989) *Justification of Formal Methods for System Specification* Software Engineering Journal 4(1) Jan 1989

Cohen91 Cohen B. (1991) *Lecture notes to Software Specification and Validation* presented at City University 9th to 13th September 1991

Cohen92 Cohen B. (1992) *Soft Systems - Hard Analysis. The Roles of Proof in Information Systems Construction* Inaugural Address delivered at City University December 1, 1992

Cohen93 Cohen B. & Molteno, B. (1993) *The Search for a Universal Model of Healthcare* Proceedings of MIE 93

Collins92 Collins, T. *The wasted millions.* Computer Weekly, 7 May 1992

CompApr92 Hayward, D. *London Ambulance places dispatch system on sick list.* Computing, 2 April 1992

CompAug92 *NHS in IT costs shock.* Computing, 20 August 1992

CompJul92 *Wessex RHA summons police to help IT inquiry.* Computing, 30 July 1992

CompNov92 *MP prescribes NHS IT audit.* Computing, 19 November 1992

Davis94 Davis, A. & Hsia P. *Giving Voice to Requirements Engineering* IEEE Software March 1994: pp 12-16.

Dearnley83 Dearnley P., and Mayhew P. (1983) *In Favour of System Prototypes and their Integration into the Systems Development Cycle* Computer Journal 26 1 pp 36-42

Dijkstra75 Dijkstra, E. (1975) *Guarded Commands, Nondeterminacy, and Formal Derivation of Programs*. Communications of the ACM, 18: pp 453-457

Disken90 Disken S, Dixon M, Halpern S, Schocket G (1990) *Models of Clinical Management.* London: IHSM.

Downs92 Downs E., Clare P., Coe I. (1992) *Structured Systems Analysis and Design Method - Application and Context* Englewood Cliffs, NJ: Prentice Hall

Doyle93 Doyle, V. C. F., Carson E, Sönksen P. *Customer Supplier Modelling as a Framework for Quality Improvement* in Malek, M., Rashquinha, J., and Vacani, P., (1993) Strategic Issues in Healthcare Management. Chichester John Wiley & Sons Ltd.

Feyer93 Feyerabend P. (1993) *Against Method* London: Verso

Flinn85 Flinn B, Sørensen I (1985) *CAVIAR: A Case Study in Specification* Oxford University Computing Laboratory Programming Research Group. Technical Monograph PRG-48

Floyd91 Floyd C., Züllighoven H., Budde R., & Keil-Slawik R. (1991) *Software Development and Reality Construction* Berlin: Springer-Verlag

Flynn93 Flynn D., and Fragoso-Diaz O. (1993) *Conceptual EuroModelling: How do SSADM and MERISE Compare?* European Journal of Information Systems **2** (3) pp 169-183

Fox92 Fox, J., *et al. LEMMA: Methods and Architectures for Logic Engineering in Medicine.* In: Noothoven van Goor, J., and Cristensen, J., (1992), *Advances in Medical Informatics: Results of the AIM Exploratory Action.* Amsterdam IOS Press.

GAO79 U.S. Government Accounting Office (1979) *Contracting for computer software development - serious problems require management attention to avoid wasting additional millions.* Technical Report FFGMSD-80-4, U.S. Government Accounting Office, November 1979

Gardiner91 Gardiner P. & Vickers T. (1991) *The Logic of B* Oxford: Oxford University Computing Laboratory Programming Research Group. Technical Monograph PRG-92.

Glykas94 Glykas M. (1994) *Agent Relationship Analysis in Organisational Transformation* PhD Thesis: Cambridge University, Department of Engineering

Goguen90a Goguen J. (1990) *Formal Methods: A Position Paper* in Goguen J. *Four Pieces on Error, Truth, and Reality* Oxford University Computing Laboratory Programming Research Group. Technical Monograph PRG-89 pp 2-10

Goguen90b Goguen J. (1990) *The Denial of Error* in Goguen J. *Four Pieces on Error, Truth, and Reality* Oxford University Computing Laboratory Programming Research Group. Technical Monograph PRG-89 pp 18-28

Goguen92 Goguen J. *The Dry and the Wet* in Proceedings of *Conference on Information Systems Concepts*, Alexandria, Egypt, 13-15 April 1992 pp 1-18

Hall90 Hall A. (1990) *Seven Myths of Formal Methods* IEEE Software. September 1990.

Hammer93 Hammer M. (1993) Re-engineering the Corporation London: Nicolas Brealey

Harrison89 Harrison J., Harvey F., & Fowler M. (1989) Framework Model Level 2 View 3 3rd Report of the EURODIABETA consortium: AIM Project No A1019

Harrison92 Harrison S, Hunter D, Marnoch G & Pollitt C (1992) *Just Managing: Power and Culture in the National Health Service*. Kent: Macmillan

Hayden68 Hayden S., Kennison J. (1968) *Zermelo Fraenkel Set Theory* Columbus, Ohio: Merrill

Hayes93 Hayes I (1993). *Specification Case Studies* New York: Prentice Hall

Heidd62 Heidegger M (1962) *Being and Time* Oxford: Blackwell

Hill94a Hill, S. *MPs attack Department of Employment over IT waste*. Computing, 10 February 1994

Hill94b Hill, S. *NAO finds fresh errors*. Computing, 17 February 1994

Hodder86 Hodder I. (1986) Reading the Past - Current Approaches to Archaeology Cambridge: Cambridge University Press

Holden93 Holden T., Glykas M., & Wilhelmij G. (1993) *Modelling the Collective Behaviour of Organisational Agents in the Petrochemical Industry Using the Agent Relationship Morphism Methodology (ARMA)* in Kilov H. & Harvey B. Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications Washington DC Sept. 1993

Holland92 Holland J, Carson E, Sönksen P, Swindells M (1992) *The Directorate Information System at St Thomas' Hospital* in Chytil M, Duru G, Van Eimeren W, Flagle Ch; editors *Health Systems - the Challenge of Change (Proceedings of the Fifth International Conference on System Science in Health Care)* Prague: Omnipress pp 1238-1241

Hull87 Hull R, King R (1987) *Semantic Database Modelling: Survey, Applications and Research Issues* ACM Computer Surveys September 1987 pp 201-259

IHSA83 Institute of Health Service Administrators (1983) *National Health Service Management Inquiry Report* London: IHSA

IMC91 *The Common Basic Specification and Proving its Worth.* Official Notice 13215/A HSSH J1714NJ

IMC92 NHS Information Management Centre (1992) *Common Basic Specification Generic Model Volumes I & II* London: NHS Management Executive

Jackson93 Jackson M. & Zave P. (1993) *Domain Descriptions* in Proceedings of *Requirements Engineering '93* Los Alamitos, California: IEEE Computer Society Press pp 56-64

Jones90 Jones C. (1990) *Systematic Software Development Using VDM* New York: Prentice Hall

Jones91 Jones, A (1991) *Non-Technological Factors and Perspectives* 10th report of the EURODIABETA Consortium: AIM Project No A1019.

Kalra93 Kalra D. & Ingram D. (1993) The Good European Health Record in Conference Proceedings HC93: Current Perspectives in Healthcare Computing 1993 Weybridge: BJHC Ltd pp 543-549

Kammer84 Kammersgaard J. *A Discussion of Prototyping Within a Conceptual Framework* in Budde R., Kuhlenkamp K., Mathiassen L., and Zullighoven H. *Approaches to Prototyping* Berlin: Springer-Verlag pp 294-321

Koestler89 Koestler A. (1989) *The Act of Creation* London: Arkana

KPMG89 KPMG Peat Marwick (1989) *St Thomas' Hospital IT Strategy Report*

Kuhn70 Kuhn, T.(1970) *The Structure of Scientific Revolution* Chicago Chicago University Press

Lakatos76 Lakatos I (1976) *Proofs and Refutations: the Logic of Mathematical Discovery* Cambridge: Cambridge University Press

Lakoff87 Lakoff G. (1987) *Women, Fire and Dangerous Things* Chicago: University of Chicago Press

Lyotard84 Lyotard J.-F. *The Postmodern condition: a Report on Knowledge*: Theory and History of Literature, Vol 10 Manchester University, 1984.

Martin89 Martin J. (1989) *Information Engineering* Englewood Cliffs, NJ: Prentice Hall

McCrack82 McCracken D., and Jackson M. (1982) *Life Cycle Concept Considered Harmful* ACM SIGSOFT Software Engineering Notes 17 02/04/1982

McNevin93 McNevin, A. *Year delay to crime system.* Computing, 16 December 1993

Morgan86 Morgan G. (1986) *Images of Organisation* Newbury Park, California: Sage

Mumford86 Mumford E. (1986) *Designing Systems for Business Success, the ETHICS Method* Manchester: Manchester Business School Publications

NAHAT91 National Association of Health Authorities and Trusts (1991) *NHS Handbook (7th Edition)* London: The Macmillan Press Ltd

NAO94 National Audit Office (1994) *Report on the Social Fund Account for 92/93* February 1994 London: HMSO

NHS83 Department of Health and Social Security (1983) *'NHS Management Inquiry'* Press Release no. 83/30, 3 February

NHS86 DHSS (1986) *Health Notice (86) 34: Resource Management (Management Budgeting) in Health Authorities* London: DHSS

NHS89 Secretaries of State for Health, Wales, Northern Ireland and Scotland (1989). Working for Patients, London, HMSO.

NHS90 Secretaries of State for Health, Wales, Northern Ireland and Scotland (1990). The National Health Service and Community Care Act (1990), London, HMSO.

NScApr92 *Computing the cost of health care.* New Scientist, 25 April 1992 pp 22 - 23

OED87 *Compact Oxford English Dictionary, 2nd Edition* (1987). Oxford: Oxford University Press pp1674

PAC93 Public Accounts Committee (1993) *Department of Employment Management of Field System* House of Commons paper 93-94 Season London: HMSO

Pack91 Packwood T, Keen J, Buxton M (1991) *Hospitals in Transition (The Resource Management Experiment)* Milton Keynes: O.U.P.

Politt91 Politt C, Harrison S, Hunter D, and Marnoch G (1991) *General Management in the NHS: The initial impact 1983-88* in Public Administration Vol 69 pp 61-83

Popper34 Popper, K (1934) *Logik der Forschung* Vienna: Springer-Verlag

Popper59 Popper, K (1959) *The Logic of Scientific Discovery* London Hutchinson & Co

Popper80 Popper K. (1980) *The Logic of Scientific Discovery* London: Unwin Hyman

Popper92 Popper, K. (1992) *Unended Quest - An Intellectual Autobiography* London Routledge

Pye88 Pye R, Bates R J, Heath L. (1988) *Profiting from office automation: Office automation pilots*. DTI, London

Rector93 The GALEN Project Consortium (Rector, A. *et al*) (1993) *GALEN: Generalised Architecture for Language Encyclopaedias and Nomenclature in Medicine* in: Commission of the European Communities DG XIII 1993 *Annual Technical Report on RTD: Health Care*

Rumbaugh91 Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W. (1991) *Object-Oriented Modelling and Design* Englewood Cliffs, NJ: Prentice Hall

Russell89 Russell, B. (1989) *A History of Western Philosophy* London Unwin

Sangh94 Sanghrajka, S. (1994) *An Outpatient Contract Management System* Computing Sciences BSc Final Year Report Surrey University

Schieber87 Schieber G (1987) *Financing and Delivering Health Care: A Comparative Analysis of OECD Countries*. Paris: Organisation of Economic Co-operation and Development Publication Service.

Schuman90 Schuman S., Pitt D., & Byers P. (1990) *Object Oriented Process Specification* University of Surrey Computing Sciences Technical Report CS-90-01

Scott91 Scott-Morton P. (1991) *The Corporation of the 1990s: Information, Technology, and Organisational Transformation* New York: Oxford University Press

Siddiqi94 Siddiqi J. *Challenging the Universal Truths of Requirements Engineering* IEEE Software March 1994: pp 18-19.

Smith92 Smith, S (1992). *Current Care Profiles* Diabetes and Endocrine Day Centre, St Thomas' Hospital

Spivey89 Spivey I (1989) *The Z Notation - A Reference Manual* New York: Prentice Hall

SWTRHA92 *Report of the Inquiry into the London Ambulance Service* (1992), South West Thames Regional Health Authority.

Tomlin92 Tomlinson B. (1992) *Report of the Inquiry into London's Health Services* London: HMSO.

Water93 Waterhouse, R (1993) *Dossier shows 'evidence' of queue-jumping* Independent Newspaper, Thursday 30 September, pp 8

Webster88 Webster C. (1988) *The Health Services Since the War*, Vol 1: Problems of Health Care - the Health Service before 1957. London: HMSO

White87 Whitehead A., & Russell, B. (1987) *Principia Mathematica (to *56)* Cambridge: Cambridge University Press

Winog87 Winograd T. & Flores F. (1987) *Understanding Computers and Cognition* Reading, Massachusetts: Addison-Wesley

Witt53 Wittgenstein L. (1953) *Philosophical Investigations Part 1* Oxford: Blackwell

Yourdon78 Yourdon E, Constantine L (1978) *Structured Design* New York: Yourdon

Yourdon89 Yourdon E (1989) *Modern Structured Analysis* Englewood Cliffs, NJ: Prentice Hall

The Requirements Analysis & Design for a Clinical Information System: A Formal Approach

by

Jeremy David Hassé Holland

A PhD Thesis

submitted to

City University,

The Department of Systems Science

January 1995

Appendices

Table of Contents

Appendix 1:

A Glossary of Set Theoretic Symbols

This glossary is intended to be used as a guide to the meaning of the set theoretic notation used in the body of the thesis and in the appendices. It is neither rigorous nor complete. For such a 'mathematically valid' definition of the terms of set theory, the reader is referred to standard algebraic reference books.

As well as being named, the semantic properties of each symbol are explained in two or three ways. Where appropriate a 'formal' definition of the symbol is given: this is either taken from the 'mathematical toolkit' in Spivey [Spivey89] or based on the style used there. An informal explanation of the symbol's meaning is given. Finally an example, or a number of examples, of the use of the symbol is provided along with any additional comments that are considered to be pertinent.

The binding order of the operators presented below is explained briefly at the back of this appendix. The normal mathematical symbols such as +, *, -, <, etc are not explained here.

Quantifiers

∀, ∃.

∀ is the Universal Quantifier, also known as 'for all'. ∃ is the Existential Quantifier, also known as 'there exists'.

Informal Definition

Both of these symbols are used in expressions where a predicate must be quantified - that is it is true under certain conditions and with certain provisos. The format of these expressions is:

∀ (or ∃) *member*: *Set* • *Predicate*(*member*).

This should be read as: For every (or far at least one) *member* of *Set*, *Predicate* is true when that *member* is provided to *Predicate* to make it a proposition.

Examples

An example of this might be as follows:

∀n: SetofAllNumbers • n + n ∈ SetofAllNumbers

or for every member of the set of all numbers, that member added to itself is also a member of the set of all numbers (the membership symbol, ∈, is explained below). Another example is

∃n: SetofAllNumbers • n > 100

or there is at least one member of the set of all numbers that is greater than 100. That there are in fact an infinity of such numbers does not invalidate the expression. Quantifiers can range over a number of members such as in the expression

∀m,n: SetofAllNumbers • m + n ∈ SetofAllNumbers

or for any pair of numbers the sum is also a number. Quantifiers can be arranged in a sequences such as in the expression

∀n: SetofAllNumbers • ∃m: SetofAllNumbers • m > n

or for any number, there is always at least one number bigger than it.

Not

~

~ can be read as 'not'.

Informal Definition

When ~ is placed in from of a predicate or proposition, we may take it to mean that that predicate or proposition is negated - it does not hold.

Examples

Negation applies to rules rather than instances, and so we expect to see it as part of predicates or propositions. For example:

$\sim A = B$

or A is not equal to B;

$\sim \exists x:X \bullet P(x)$

or there is no x for which P of x is true;

$\sim \forall x:X \bullet P(x)$

or it is not true that P(x) always holds;

$\forall x:X \bullet \sim P(x)$

or P(x) is never true, no matter what x is; and

$\exists x:X \bullet \sim P(x)$

or there is some x for which P(x) does not hold.

The second and fourth of these examples are equivalent, as are the third and fifth.

Set Enumeration Braces

{}

Informal Definition

These symbols are used to define a set by the enumeration of its contents. The discrete expressions separated by commas (and not a component part of such a discrete expression) within the braces are members of the set indicated by those braces.

Examples

The set of all positive integers less than 5 could be written as

{1, 2, 3, 4}.

As there is no ordering over a simple set, we could equally have written

{2, 3, 1, 4}.

A set might be a discrete expression so the expression

{{1, 2}, 1, 2, 3, 4}

represents that set which consists of the elements: the set {1, 2}; the number 1; the number 2; the number 3; and the number 4.

Membership

∈, ∉

∈ can be read as 'is a member of'. ∉ can be read as 'is not a member of'.

Informal Definition

The ∈ symbol occurs in expressions which take the form

s ∈ S.

This means that the expression to the left of the symbol, (in this case s) is a member of the set indicated by the expression to the right of the symbol (in this case S).

The ∉ symbol occurs in expressions which take the form

s ∉ S.

This means that the expression to the left of the symbol, (in this case s) is <u>not</u> a member of the set indicated by the expression to the right of the symbol (in this case S).

Examples

Suppose we call the set {{1, 2}, 1, 2, 3, 4} X.

We can then say

1 ∈ X,
5 ∉ X, and
{1, 2} ∈ X.

The Null Set

∅

This symbol can be read as 'the null set', 'null', or 'the empty set'.

Informal Definition

The null set is the set which has no members.

Examples

The set of all numbers that are both greater than 5 and less than 5 is an example of the null set. It is used in similar expressions indicating that no member of a set can obey a particular predicate. Because the set has no members we can say

∀x: X • x ∉ ∅

where X is an arbitrary set. There are no members of any set that are members of the null set.

The null set is sometimes written as

{}.

Pair (or Tuple)

()

Discrete expressions within brackets can (sometimes) represent an ordered pair, or if there are more than two of them, a tuple.

Informal Definition

An ordered pair is a quantity that consists of two discrete expressions, separated by a comma. It is like a two-membered set except that the order of the elements is important. If there are more than two members then the quantity is called a triple, quadruple, pentuple, or more generally tuple or n-tuple.

Examples

A pair is most commonly used to talk about members of relations (see below). Thus the pairs

(1, 2), (6, 8), (2, 3), and (4, 9) are all members of the set <.

Binary Conjunctions: And, Or, & Implies

$\wedge$, $\vee$, $\Rightarrow$.

$\wedge$ can be read as 'and'. $\vee$ can be read as 'or'. $\Rightarrow$ can be read as 'implies' (although this can sometimes be misleading).

Informal Definition

These symbols are conjunctions which can join two predicates or propositions. The expression $P_1 \wedge P_2$ means both predicate P_1 and P_2 hold. The expression $P_1 \vee P_2$ means that either predicate P_1 or predicate P_2 (and possibly both) hold. The expression $P_1 \Rightarrow P_2$ means that if P_1 holds then so does P_2: if P_1 does not hold, then we can say nothing about P_2.

Examples

6 is divisible by both 3 and 2. We can state this as follows:

$6 / 3 \in \mathbb{N} \wedge 6 / 2 \in \mathbb{N}$ (see below for description of $\mathbb{N}$).

Any number is odd or even. We can state this as follows:

$\forall n: \mathbb{N} \bullet n / 2 \in \mathbb{N} \vee (n+1) / 2 \in \mathbb{N}$.

Any prime is divisible only by itself and 1:

$\forall p: \mathbb{P} \bullet \forall n: \mathbb{N} \bullet p / n \in \mathbb{N} \Rightarrow n = p \vee n = 1$ (where P is the set of primes).

This says that for any prime, and for any number, the divisibility of the prime by the number implies that the number is either the prime or one. The reason why it is sometimes confusing to consider $\Rightarrow$ as 'implies' is that the following expressions are both true:

$1 + 1 = 3 \Rightarrow 1 + 1 = 5$

and

$1 + 1 = 3 \Rightarrow 1 + 1 = 2.$

This is because the expression to the left of the implication symbol is false, so we cannot say anything about the expression to the right.

Cartesian Product

x

Informal Definition

This symbol is generally found in expressions of the type A x B where A and B are (expressions which refer to) sets. A x B is a set of pairs such that for any member of A, and for any member of B, there is one pair in the product set where the member of A is paired up with the member from B.

Examples

Suppose A was the set {a1, a2, a3} and B the set {b1, b2}.

A x B is then the set

{(a1, b1), (a1, b2), (a2, b1), (a2, b2), (a3, b1), (a3, b2)}.

Square

2

Formal Definition

$X^2 == X \times X$

Informal Definition

The square of a set is the cartesian product of the set with itself.

Examples

Taking the set A from the previous example, then A2 can be enumerated as follows:

{(a1, a1), (a1, a2), (a1, a3), (a2, a1), (a2, a2), (a2, a3), (a3, a1), (a3, a2), (a3, a3)}.

In general, a set of pairs where all the elements of each pair are taken from the same set is called a graph.

Set Constructor

Set[]

The set constructor applied to a set is sometimes called the 'power set' of the set.

Formal Definition

$(\forall S: Set[SS] \bullet \forall s: S \bullet s \in SS) \wedge (\sim\exists S \notin Set[SS] \bullet \forall s: S \bullet s \in SS)$

Informal Definition

The set constructor applied to, or power set of, a set, S, is a set of sets. Each of the sets in Set[S] is a subset of S - that is, every element of any set in Set[S] is also an element of S. In short, Set[S] is the set of all subsets of S.

Examples

If we take A as before, then Set[A] is

{{a1, a2, a3}, {a1, a1}, {a1, a2}, {a1, a3}, {a2, a2}, {a2, a3}, {a3, a3}, {a1}, {a2}, {a3}, ∅}.

Union, Intersection, and Difference

∪, ∩, \

The symbol ∪ is called 'union'. The symbol ∩ is called 'intersection'. The symbol \ is called 'set difference'.

Formal Definition

[X]

$$\_\cup\_, \_\cap\_, \_\setminus\_ : Set[X] \times Set[X] \rightarrow Set[X]$$

$$\forall S, T : Set[X] \bullet$$
$$S \cup T = \{x : X \mid x \in S \vee x \in T\} \wedge$$
$$S \cap T = \{x : X \mid x \in S \wedge x \in T\} \wedge$$
$$S \setminus T = \{x : X \mid x \in S \wedge x \notin T\}$$

Informal Definition

These symbols normally occur in expressions such as A ∪ B, A ∩ B, or A \ B where A and B are (expressions which refer to) sets. A ∪ B, or the union of A and B is a set which contains all the members of A and all the members of B and no more. A ∩ B, or the intersection of A and B, is a set which contains all the members of A that are also members of B. A \ B, or the set difference of A and B, is a set which contains all the members of A which are not in B.

Examples

Let us take the following sets:

A = {a1, a2, a3, a4, a5}

and

B = {a2, a4, a6, a7}

then

A ∪ B = {a1, a2, a3, a4, a5, a6, a7},
A ∩ B = {a2, a4}, and
A \ B = {a1, a3, a5}.

Subset, Superset, and Proper Subset

$\subseteq, \supseteq, \subset$.

$\subseteq$ can be read as 'is a subset of'. $\supseteq$ can be read as 'is a superset of'. $\subset$ can be read as 'is a proper subset of'.

Formal Definition

[X]

$\_\subseteq\_, \_\supseteq\_, \_\subset\_ : Set[X] \leftrightarrow Set[X]$

$\forall S, T: Set[X] \bullet$
$(S \subseteq T \equiv (\forall x: X \bullet x \in S \Rightarrow s \in T)) \wedge$
$(S \supseteq T \equiv (\forall x: X \bullet x \in T \Rightarrow s \in S)) \wedge$
$(S \subset T \equiv S \subseteq T \wedge S \neq T)$

Informal Definition

If $A \subseteq B$ then every member of the set A is also a member of the set B. If $A \supseteq B$, then every member of b is a member of A. If $A \subset B$ then every member of the set A is also a member if the set B, and the two sets are not equal - there must be at least one member of B that is not a member of A.

Examples

Suppose we had three sets, X, Y, and Z, where

$X = \{a1, a2, a3, a4, a5\}$,
$Y = \{a1, a2, a3\}$, and
$Z = \{a1, a2, a3\}$.

We can say that

$Y = Z$,
$Y \subseteq X$,
$X \supseteq Y$,
$Y \subseteq Z$, and
$Y \subset X$, but not
$Y \subset Z$

as there is not at least one element of Z that is not in Y.

Distributed Union and Intersection

$\cup, \cap$.

$\cup$ is called the 'distributed union'. $\cap$ is called the 'distributed intersection'.

Formal Definition

[X]

$\cup, \cap : Set[Set[X]] \rightarrow Set[X]$

$\forall A: Set[Set[X]] \bullet$

$$\cup A = \{x: X \mid \exists S: A \bullet x \in S\} \wedge$$
$$\cap A = \{x: X \mid \forall S: A \bullet x \in S\}$$

Informal Definition

The distributed union and intersection are operators which can be applied to sets of sets. The returned quantity is a set that is the union of each of the sets in the set of sets.

Examples

Suppose we have a set

$X = \{\{a1, a2, a3, a4\}, \{a1, a3, a5, a6\}, \{a2, a3, a5, a7\}\}$

then

$\cup X = \{a1, a2, a3, a4, a5, a6, a7\}$ and
$\cap X = \{a3\}$.

Of course, for any set

$\cup Set[X] = X$

as each set in Set[X] is a subset of X, including X itself and

$\cap Set[X] = \varnothing$

as Set[X] includes the null set.

Relation

$\nrightarrow$, $\rightarrow$

$\nrightarrow$ generates a set of partial relations. $\rightarrow$ generates a set of total relations.

Formal Definition

$X \nrightarrow Y == Set[X \times Y]$

$X \rightarrow Y == \{r: Set[X \times Y] \mid \forall x: X \bullet \exists y: Y \bullet (x,y) \in r\}$

Informal Definition

$X \nrightarrow Y$ is the set of all subsets of the cartesian product of X and Y. $X \rightarrow Y$ is a subset of this - that set of all subsets of the cartesian product of X and Y where for each subset, every member of X is the first component of an element of that subset. Note that $X \nrightarrow Y$ is not a partial relation - rather it enables us to generate them. For this reason it is most commonly used to declare types as in

$P\_Rel: X \nrightarrow Y$

or P_Rel is an element of the set of partial relations, and is thus a partial relation itself - a subset of $X \times Y$.

Examples

Suppose we have the sets

A = {a1, a2, a3, a4} and
B = {b1, b2, b3}

then if we say that

P_Rel: A ⇻ B and
T_Rel: A ↔ B

then the following are possible values of P_Rel

{(a1, b1), (a2, b2), (a3, b3)}
{(a1, b1), (a2, b1), (a3, b1)}
∅

and the following possible values of T_Rel

{(a1, b1), (a2, b2), (a3, b3), (a4, b4)}
{(a1, b1), (a2, b1), (a3, b1), (a4, b1)}.

Note that A ↔ B ⊆ A ⇻ B thus any set that is a possible value of T_Rel is also a possible value of P_Rel

Domain and Codomain

Dom(), Cod()

Dom(R) is referred to as the domain of relation R. Cod(R) is referred to as the codomain of relation R.

Formal Definition

[X, Y]

Dom: (X ↔ Y) → Set[X]
Cod: (X ↔ Y) → Set[Y]

∀R: X ↔ Y •
Dom(R) = {x: X; y: Y | (x, y) ∈ R • x} ∧
Cod(R) = {x: X; y: Y | (x, y) ∈ R • y}

Informal Definition

The domain of a relation is the set of all elements which form the first component of the pairs that are the members of the relation. The codomain of a relation is the set of all elements which form the second component of the pairs that are the members of the relation.

Examples

Suppose we have relations with the following values

R1 = {(a1, b1), (a2, b2), (a3, b3), (a4, b4)} and
R2 = {(a1, b1), (a2, b1), (a3, b1), (a4, b1)}

then

Dom(R1) = Dom(R2) = {a1, a2, a3, a4}
Cod(R1) = {b1, b2, b3, b4}
Cod(R2) = {b1}.

We can say for all relations, total or partial, that

R: Cod(R) $\leftrightarrow$ Dom(R).

Function

$\nrightarrow$, $\rightarrow$.

$\nrightarrow$ generates a set of partial functions. $\rightarrow$ generates a set of total functions.

Formal Definition

$$X \nrightarrow Y == \{f: X \leftrightarrow Y \mid \forall x: X; y1, y2: Y \bullet (x, y1) \in f \land (x, y2) \in f \Rightarrow y1 = y2\}$$

$$X \rightarrow Y == X \nrightarrow Y \cap X \leftrightarrow Y$$

Informal Definition

$X \nrightarrow Y$ is the set of all relations from X to Y where no two elements in the relation's domain share a member of Y as their partner (in other words, the number of elements in the domain of the relation is the same as the number in the codomain). $X \rightarrow Y$ is the set of all total relations from X to Y that are also functions.

Because each element of the domain of a function is paired up with exactly one element in the codomain, we can use the expression F(a) to refer to that element of the codomain of F that is paired up with the element a.

Examples

Suppose we have the sets

A = {a1, a2, a3, a4} and
B = {b1, b2, b3}

then if we say that

P_Fn: $A \nrightarrow B$ and
T_Fn: $A \rightarrow B \subseteq$

then the following are possible values of P_Fn

{(a1, b1), (a2, b2), (a3, b3)}
{(a1, b3), (a2, b2), (a3, b1)}
$\varnothing$

and the following possible values of T_Fn

{(a1, b1), (a2, b2), (a3, b3), (a4, b4)}
{(a1, b4), (a2, b3), (a3, b2), (a4, b1)}.

We can construct an inclusion hierarchy for all the relations described.

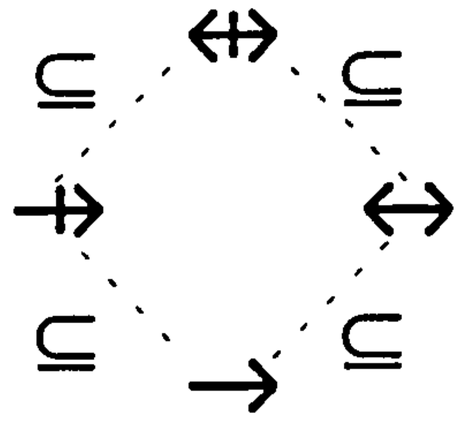


Figure App1-1: Inclusion hierarchy for total function, partial function, total relation, and partial relation generation operators.

Inverse

-1

R^{-1} is referred to as the inverse of the relation R.

Formal Definition

[X, Y]

$_^{-1}: (X \leftrightarrow Y) \rightarrow (Y \leftrightarrow X)$

$\forall R: X \leftrightarrow Y \bullet$
$R^{-1} = \{x: X; y: Y \mid (x,y) \in R \bullet (y,x)\}$

Informal Definition

An inversion of a relation is that relation with the order of the consituent pairs reversed.

Examples

Suppose we have relations with the following values

R1 = {(a1, b1), (a2, b2), (a3, b3), (a4, b4)}, and
R2 = {(a1, b1), (a2, b1), (a3, b1), (a4, b1)}.

Then

$R1^{-1}$ = {(b1, a1), (b2, a2), (b3, a3), (b4, a4)}, and
$R2^{-1}$ = {(b1, a1), (b2, a1), (b3, a1), (b4, a1)}.

Identity Constructor

id[]

id[X] is the identity function for the set X.

Formal Definition

$id[X] == \{x: X \bullet (x,x)\}$

Informal Definition

The identity function of a set is that function created by pairing every member of the set up with itself.

Examples

Suppose we have the sets

A = {a1, a2, a3, a4} and
B = {b1, b2, b3}

then

id[A] = {(a1, a1), (a2, a2), (a3, a3), (a4, a4)}, and
id[B] = {(b1, b1), (b2, b2), (b3, b3)}.

These are examples of graphs. A graph is a relation where both the domain and the codomain are taken from the same set. A tree is a graph that is also a function.

Relational Image

im

(im R) A is the relational image of the set A through the relation R.

Formal Definition

[X, Y]

$(\text{im } \_) \_ : (X \leftrightarrow Y) \times \text{Set}[X] \rightarrow \text{Set}[Y]$

$\forall R: (X \leftrightarrow Y); S: \text{Set}[X] \bullet$
$(\text{im } R)\ S = \{x:X; y:Y \mid x \in S \wedge (x,y) \in R \bullet y\}$

Informal Definition

The relational image of a set A through a relation R is the largest subset of the codomain of R such that every member of that subset is the second part of a pair from R whose first part is a member of A.

Examples

Suppose we have the relation

R = {(a1, b1), (a1, b2), (a2, b2), (a3, b2), (a4, b3)}

and the sets

W = {a1, a2, a4},
X = {a1, a2},
Y = {a1}, and
Z = {a2}.

The following are the relational images of the above sets through R - that is:

(im R) W = {b1, b2, b3};
(im R) X = {b1, b2};
(im R) Y = {b1, b2}; and
(im R) Z = {b2}.

Backward Relational Composition

o

R1 ° R2 is the backward relational composition of relations R1 and R2.

Formal Definition

$$[X]$$

$$\_ \circ \_ : (Y \leftrightarrow Z) \times (X \leftrightarrow Y) \rightarrow (X \leftrightarrow Z)$$

$$\forall R: Y \leftrightarrow Z; S: X \leftrightarrow Y \bullet R \circ S = \{x:X; y:Y; z:Z \mid (y,z) \in R \land (x,y) \in S \bullet (x,z)\}$$

Informal Definition

The backward relational join of two relations, R and S (R ° S), is another relation, T, such that for every pair of pairs from R and S where the first part of R is the same as the second part of S, T contains the first part of S followed by the second part of R.

Examples

Suppose we had two relations Age and Height where

Age = {(Robert, 8), (Richard, 25), (Rowan, 1), (Ryan, 66), (Rapunzel, 28)}, and
Height = {(Robert, 4' 3"), (Richard, 6' 2"), (Rowan, 3' 0"), (Rapunzel, 5' 5"), (Radovan, 5' 8")}.

Now there are no pairs from Age and Height such that the first part of the pair from Age is the same as the second part of pair from Height. If, however, we take the second part of the inverse of Height we find that there are matching components.

Thus where

$Height^{-1}$ = {(4' 3", Robert), (6' 2", Richard), (3' 0", Rowan), (5' 5", Rapunzel), (5' 8", Radovan)}

we can say

Age ° $Height^{-1}$ = {(4' 3", 8), (6' 2", 25), (3' 0", 1), (5' 5", 28)}.

Relational Join

◊

R1 ◊ R2 is the relational join of two relations R1and R2.

Formal Definition

$$[X, Y, Z]$$

$$\_ \lozenge \_ : (X \leftrightarrow Y) \times (X \leftrightarrow Z) \rightarrow (X \leftrightarrow (Y \times Z))$$

$$\forall R1:(X \leftrightarrow Y); R2:(X \leftrightarrow Z) \bullet R1 \lozenge R2 = \{x:X; y:Y; z:Z \mid (x,y) \in R1 \land (x,z) \in R2 \bullet (x,(y,z))\}$$

Informal Definition

The relational composition of two relations, R and S, is another relation T. R, S, and T all have domains of the same type. The codomain of T is a subset of the cartesian product of the codomains of R and S. T is constructed in the following way. Where a pair from R has the same first part as a pair from S, that first part is paired up eith a pair made from the second part of the member from R followed by the second part of the member from S.

Examples

Suppose we have the two sets from the previous example:

Age = {(Robert, 8), (Richard, 25), (Rowan, 1), (Ryan, 66), (Rapunzel, 28)}, and
Height = {(Robert, 4' 3"), (Richard, 6' 2"), (Rowan, 3' 0"), (Rapunzel, 5' 5"), (Radovan, 5' 8")}

then

Age ◊ Height = {(Robert, (8, 4' 3")), (Richard, (25, 6' 2")), (Rowan, (1, 3' 0")), (Rapunzel, (28, 5' 5"))}.

Domain & Codomain Restrict & Subtract

◁, ▷

A ◁ R is the domain restriction of relation R by set A. R ▷ A is the codomain restriction of relation R by set A.

Formal Definition

[X, Y]

$$\_\triangleleft\_ : \mathrm{Set}[X] \times (X \leftrightarrow Y) \rightarrow (X \leftrightarrow Y)$$

$$\_\triangleright\_ : (X \leftrightarrow Y) \times \mathrm{Set}[Y] \rightarrow (X \leftrightarrow Y)$$

$$\forall S:\mathrm{Set}[X];\ T:\mathrm{Set}[Y];\ R:(X \leftrightarrow Y) \bullet$$
$$S \triangleleft R = \{x:X;\ y:Y \mid x \in S \wedge (x,y) \in R \bullet (x,y)\} \wedge$$
$$R \triangleright T = \{x:X;\ y:Y \mid (x,y) \in R \wedge y \in T \bullet (x,y)\}$$

Informal Definition

A ◁ R is the largest subset of relation R such that each member of its domain is also a member of the set A. R ▷ A is the largest subset of relation R such that each member of its codomain is also a member of the set A.

Examples

Suppose we have the relation

R = {(a1, b1), (a2, b1), (a3, b2), (a3, b3), (a4, b4)}

and the sets

W = {a1, a2, a3};
X = {b1, b3};
Y = {a1, a3, a5, a7}; and
Z = ∅

then we can construct the follwing relations:

$W \lhd R = \{(a1, b1), (a2, b1), (a3, b2), (a3, b3)\}$;
$R \rhd X = \{(a1, b1), (a2, b1), (a3, b3)\}$;
$Y \lhd R = \{(a1, b1), (a3, b2), (a3, b3)\}$;
$Y \lhd (R \rhd X) = (Y \lhd R) \rhd X = Y \lhd R \rhd X = \{(a1, b1), (a3, b3)\}$; and
$Z \lhd R = R \rhd Z = \varnothing$.

Transitive and Reflexive Transitive Closure

$^+, ^*$

G^+ is the transitive closure of the graph G (remember that a graph is a relation where the domain and codomain are of the same type). G^* is the reflexive transitive closure of the graph G.

Formal Definition

$[X]$

$$_^+; _^*: (X \leftrightarrow X) \rightarrow (X \leftrightarrow X)$$

$$\forall G: X \leftrightarrow X \bullet$$
$$G^+ = \cap \{Q: X \leftrightarrow X \mid G \subseteq Q \wedge Q \circ Q \subseteq Q\} \wedge$$
$$G^* = \cap \{Q: X \leftrightarrow X \mid id[Dom(G) \cup Cod(G)] \subseteq Q \wedge G \subseteq Q \wedge Q \circ Q \subseteq Q\}$$

Informal Definition

The transitive closure of a graph G is the smallest graph of the same type which is a superset of G and is transitive. A transitive graph is one where for any a,b,c in the set from which G is generated, if (a,b) and (b,c) are members of G, then so also is (a,c).

The reflexive transitive closure of a graph G is the union of the transitive closure of G with the set of pairs of identical elements which are either members of the domain of G, or of the codomain of G, or of both.

Examples

Suppose we have the graphs

$G1 = \{(a1, a2), (a1, a4), (a2, a3), (a3, a4), (a3, a5)\}$ and
$G2 = \{(b1, b2), (b1, b4), (b2, b3), (b3, b4), (b5, b6), (b6, b8), (b6, b7)\}$

then we would have the following closures:

$G1^+ = \{(a1, a2), (a1, a3), (a1, a4), (a1, a5), (a2, a3), (a2, a4), (a2, a5), (a3, a4), (a3, a5)\}$;
$G1^* = \{(a1, a2), (a1, a3), (a1, a4), (a1, a5), (a2, a3), (a2, a4), (a2, a5), (a3, a4), (a3, a5), (a1, a1), (a2, a2), (a3, a3), (a4, a4), (a5, a5)\}$; and
$G2^+ = \{(b1, b2), (b1, b3), (b1, b4), (b2, b3), (b2, b4), (b3, b4), (b5, b6), (b5, b7), (b5, b8), (b6, b7), (b6, b8)\}$.

Using the graphical notation employed in the thesis we can represent the closures in the following pictorial way:

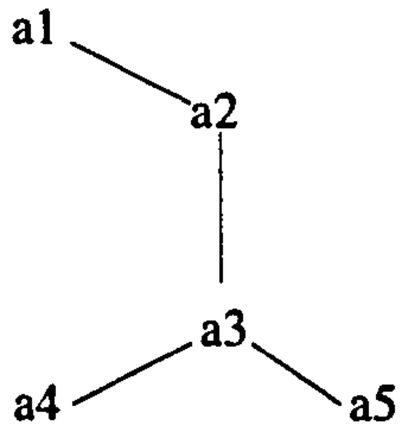
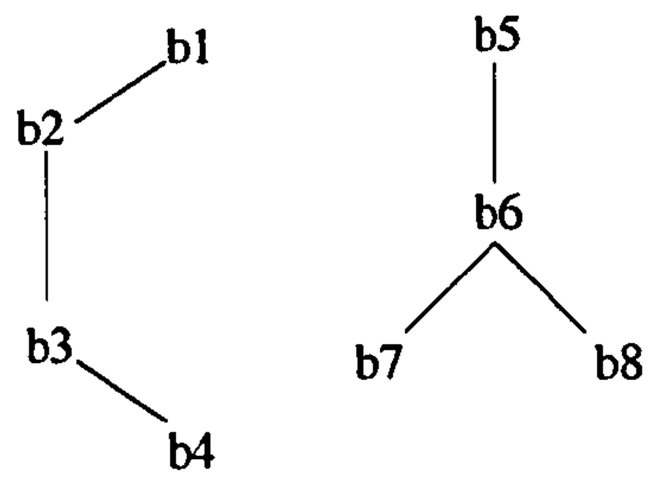
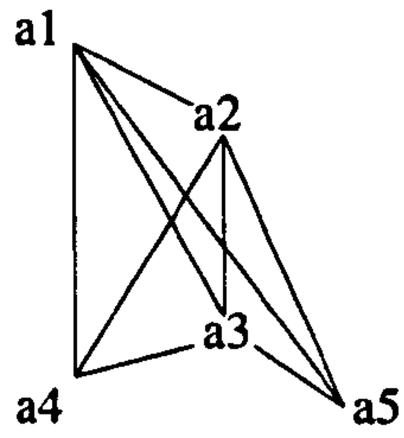
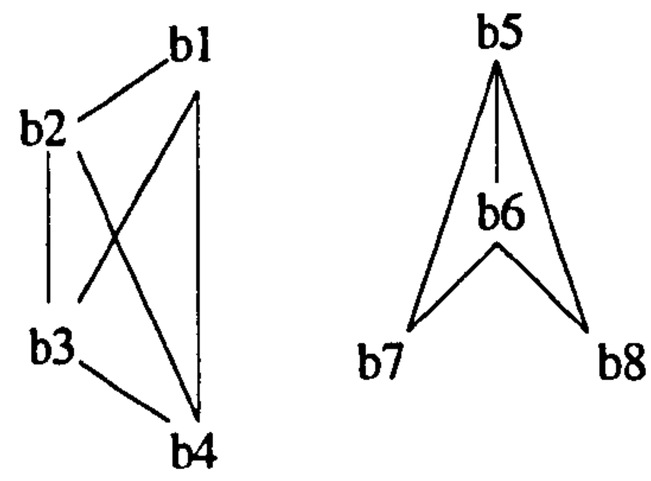
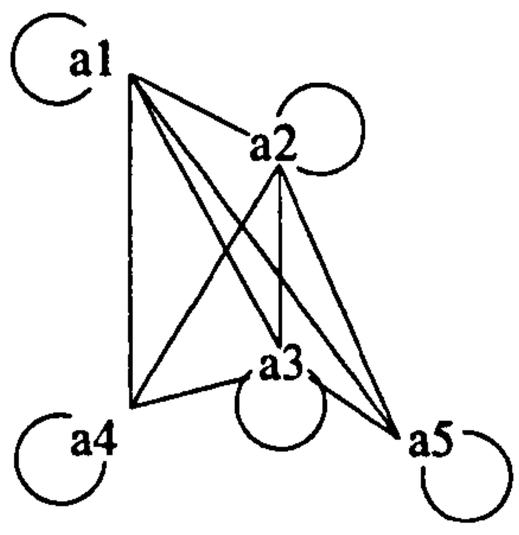
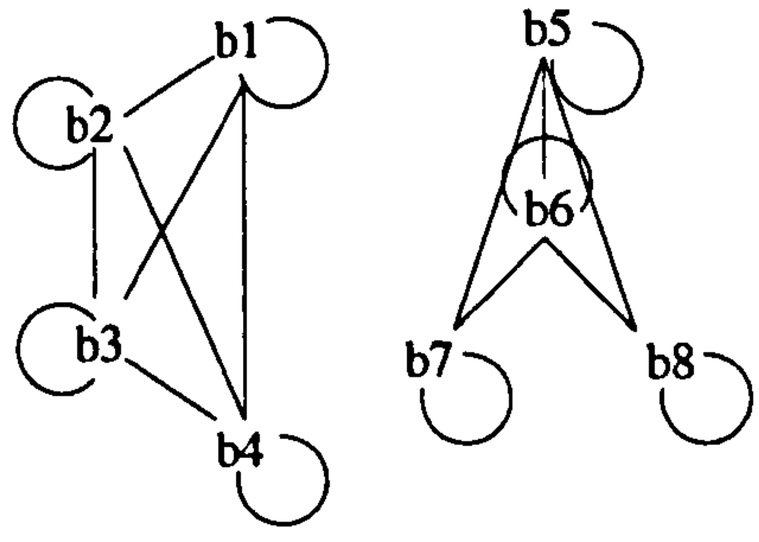
	G1	G2
G*n*		
G*n*$^+$		
G*n**		

Figure App1-2: Representation of Transitive and Reflexive-Transitive Closures

Numbers

N, N$^+$

N symbolises the set of natural numbers. N$^+$ symbolises the set of non-zero natural numbers.

Informal Definition

N is the set of all natural numbers: that is the positive integers and 0. N$^+$ is the same set, but without 0.

Cardinality

#

#A is the cardinality of set A.

Informal Definition

The cardinality of a set is the number of distinct members contained within it.

Examples

Suppose we have the sets:

A = {a, b, c, d};
B = {{a, b}, {a, b, c, d}, {a}, ∅}; and
C = ∅

then

#A = 4,
#B = 4, and
#C = 0.

Operator Precedence

When reading the set theoretic constructions described in the thesis, it should be noted that the following binding convention has been used: that set theoretic operators bind more tightly than logical operators.

For example

$A = B \cup C$

should be understood as

$A = (B \cup C)$

and not (the meaningless)

$(A = B) \cup C$.

In general there is no ambiguity as the expressions will only be well formed formulae with respect to type when they are parsed in the intended way.

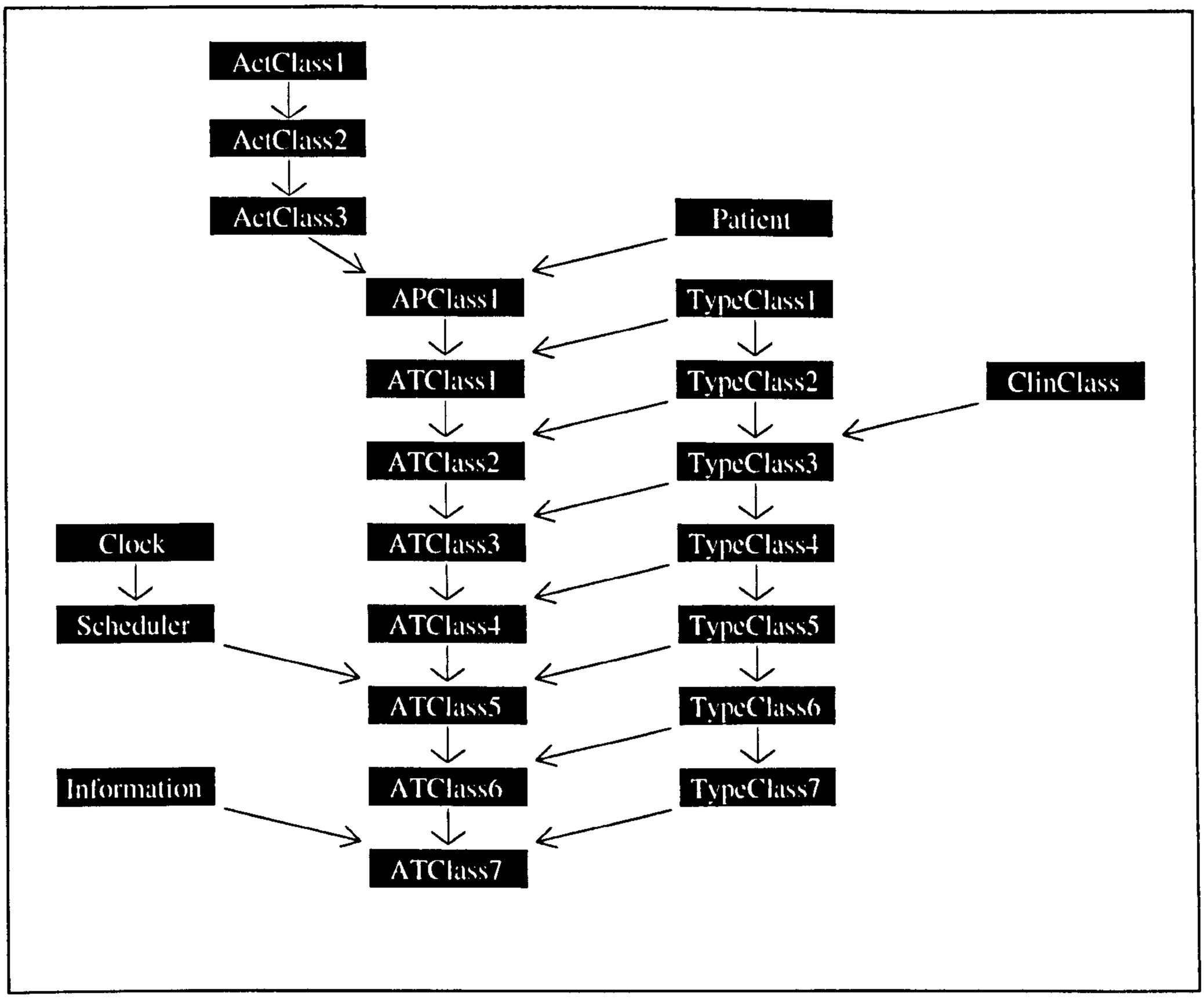
Appendix 2:

The Domain Theory

This appendix records the domain theory in its latest manifestation (the 21st recorded iteration in this form). The theory is presented in Schuman-Pitt format. Each of the type declarations, invariants, preconditions and postconditions is numbered separately. Each of the numbered predicates and abbreviations is described, and where appropriate commented on, following the schema in which it appears.

The introduction of each class can be found on the following pages

The class heirarchy of the theory.is summarised in the following table:



ActClass1

T1: Request, Proceed, Complete, In, Out, Activities: Set[A]
I1: Request ∪ Proceed ∪ Complete = In ∪ Out = Activities
I2: Request ∩ Proceed = Proceed ∩ Complete = Complete ∩ Request = In ∩ Out = ∅
Activities' = ∅

- Inherit the type declarations and invariants of the class Constants (in which the carrier sets would be defined).

Types:

- Activities is the set of patient encounters at (whatever level of generalisation) that we are concerned with. Request is the set of requested activities that have not started or have been suspended. Proceed is the set of activities that have been started but not since been suspended, and not completed. Complete is the set of completed activities. In is the set of activites that can be considered to be in the realm of the organisation we are considering. Out is the set of activities that are outside the realm of the organisation being considered.

Invariants:

- Request, Proceed and Complete are disjoint partitions of the activities set
- In and Out are disjoint partitions of the activities set

NB The Out set is different from the In set conceptually. Whereas the In set is a record of the state of the organisation in terms of which activities are currently in which status, the Out set is a set of phantom activities that record our knowledge of activities proceeding in the outside world. If an activity is not recorded in Out, it does not mean it does not exist, merely that we do not know (or care) about it. On the other hand, all the internal activitiea are recorded in In. Similarly, the set In ∩ Request is all internal activities that have been created but have not started. The set Out ∩ Request is all external activities that we are aware of and either we know they have not been started, or we do not know their status - we know they must have been created at some point, possibly by us.

ActClass1.InRequest(a)

Pr 1: a: A\Activities
Po 1: a ∈ Activities' ∩ In'

Preconditions:

- a is not yet an activity

Postconditions:

- a is now an internal activity

ActClass1.SuddenStart(a)

Pr 2: a: A\Activities
Po 2: a ∈ In' ∩ Proceed'

This operation is a primitive that enables us to model unplanned activities that are created and started at the same time.

Preconditions:

- a is not yet an activity

Postconditions:

- a is an activity internal to the organisation, and has started.

ActClass1.Start(a)

Pr 3: a: In ∩ Request
Po 3: a ∈ Proceed'

Preconditions:

- a is a request that is internal to the organisation

Postconditions:

- a is in proceed.

Note that the semantics of the Schuman-Pitt notation mean that in this case, if the activity starts the operation in the set In, it will finish the operation in the set In unless this causes an invariant to be contravened. The activity a being in In ∩ Proceed does not contravene any invariants, so a remains in In.

ActClass1.Suspend(a)

Pr 4: a: In ∩ Proceed
Po 4: a ∈ Request'

This operation enables activities to be interrupted without their being finished. Once they are interrupted, or suspended, they become requests again, waiting to be (re) started

Preconditions:

- a is a proceeding activity that is internal to the organisation

Postconditions:

- a is moved back to Request.

ActClass1.End(a)

Pr 5: a: In ∩ Proceed
Po 5: a ∈ Complete'

Preconditions:

- a is a proceeding activity that is internal to the organisation

Postconditions:

- a is moved to Complete.

ActClass1.Cancel(a)

Pr 6: a: Request
Po 6: a ∉ Activities'

Preconditions:

- a is a request (either internal or external to the organisation)

Postconditions:

- a is no longer in Activities: it remains in its carrier set A, however.

ActClass1.OutRequest(a)

Pr 7: a: A\Activities
Po 7: a ∈ Out' ∩ Request'

The .OutXxxxxx operations represent an 'update of the organisation's perceived environment'. They are not operations that create or manipulate 'Out' activities - they are operations that update our understanding of the state of these activities. Thus, for example the OutRequest operation represents the organisation finding out about the existence of a request for an activity external to the organisation.

Preconditions:

- a is not yet an activity

Postconditions:

- a is a Request that is external to the organisation

ActClass1.OutProceed(a)

Pr 8: a: (A\Activities) ∪ (Out ∩ Request)
Po 8: a ∈ Proceed'

Preconditions:

- a is either not yet an activity or is an external request.

Postconditions:

- a is a Request that is external to the organisation

ActClass1.OutComplete(a)

Pr 9: a: (A\Activities) ∪ (Out\Complete)
Po 9: a ∈ Complete'

Preconditions:

- a is either not yet an activity or is a request external to the organisation, or is a proceeding activity external to the organisation

Postconditions:

- a is a Request that is external to the organisation

◇◇◇

- Any activity that is after a complete or proceeding activity is complete
- An external activity cannot be after any activity. This reflects the lack of control we have over external activities: we cannot dictate that an external activity waits until a suitable moment - the activity will commence when its own organisation is able and wants it to start.

ActClass2.InRequest(A_b,a)

Pr 10: A_b: Set(Activities)
Pr 11: **ActClass1.InRequest**(a)
Po 10: {a} X $A_b \subseteq$ After'

Preconditions:

- A_b is a set of activities that take are to take place Before the requested activity a.
- Inherit the preconditions and postconditions of the Request operation.

Postconditions:

- a is After all activities in A_b.

ActClass2.SuddenStart(a)

If an operation is identical to one in the preceeding class in the class heirarchy, only its name is presented - the preconditions and postconditions being the same as for the operation of the same name in the earlier class.

ActClass2.Start(a)

Pr 12: (im After) {a} $\subseteq$ Complete
Pr 13: **ActClass1.Start**(a)

Preconditions:

- All activities After a must already be complete.
- Inherit the preconditions and postconditions of the Start operation

- **ActClass2.Suspend**(a)
- **ActClass2.End**(a)
- **ActClass2.Cancel**(a)
- **ActClass2.OutRequest**(a)

NB Note that there are no 'before' activities that a must succeed here. This is to preserve the invariant.

- **ActClass2.OutProceed**(a)
- **ActClass2.OutComplete**(a)

ActClass3

ActClass2

T 3: During, Includes: Activities ⇿ Activities

I 7: $Includes^{-1} = During$

I 8: $During^{+} \cap id[Activities] = \varnothing$

I 9: (im Includes) Complete ⊆ Complete

I 10: (im During) Proceed ⊆ Proceed

I 11: (After ∪ Before) ▷ Dom(During) ⊆ Includes ⚬ During

I 12: During ∩ Out X Out = ∅

- Inherit the type declarations and invariants of the class ActClass2

Types:

- Includes and During are both graphs over activities. If an activity a is included in an activity b, then a is a part of b: in delivering the service implied by b, it was decided somewhere that a must also be delivered. The concept implies medical containment and delegated 'responsibility'.

Invariants:

- Includes in the inverse of During.
- During (and hence Includes) is a DAG
- All activities Included in activities in Complete are also in Complete
- All activities During activities in Proceed are also in Proceed.
- Any two activities that are in the After relation where at least one is Included in another activity have a parent in common.
- An external activity can be part of an internal activity, or an internal activity can be part of an external activity, but an external activity cannot be part of an external activity. This reflects our knowledge rather than some objective truth. We do not know or care about the structure of activities delivered by other departments - only their names, and whether one of our activities is a part of one of their's (as in shared care) or one of their's is a part of one of ours (as in certain blood tests).

ActClass3.Create(A_b,a_n)

Pr 14: **ActClass2.InRequest(A_b,a_n)**

ActClass3.Embed(A_p,A_b,a_c)

Pr 15: A_p: Set[Activities\Complete]
Pr 16: $A_p \cap In \neq \varnothing$
Pr 17: $A_b \subseteq Dom(During \rhd A_p)$
Pr 18: **ActClass2.InRequest**(A_b,a_n)
Po 11: $\{a_c\} \times A_p \subseteq During'$

Preconditions:

- A_p is a set of activities in which the new activity is to be included. None of A_p is complete
- One of A_p must be an internal activity. If we embed an internal activity in an external activity, then there must be a 'spouse' including activity that is internal. This is because we do not care about the activity our 'top level' activity is embedded in - anything else delivered by us must be a part of our care somehow (though as in the case of shared care it need not only be embedded in one of our activities).
- Each of A_b must be embedded in one of A_p.
- Inherit the preconditions and postconditions of the Request operation.

Postconditions:

- a_c (child activity) is included in each of A_p.

ActClass3.SuddenStart(A_p,a_c)

Pr 19: A_p: Activities\Complete
Pr 20: $A_p \cap In \neq \varnothing$
Pr 21: **ActClass2.SuddenStart**(A_b,a_n)
Po 12: $\{a_c\} \times A_p \subseteq During'$
Po 13: $During'^{+} \{a_c\} \subseteq Proceed'$

Preconditions:

- A_p is a set of activities that have not been completed.
- At least one of A_p is an internal activity
- Inherit the preconditions and postconditions of the SuddenStart operation

Postconditions:

- a_c is included in each of A_p.
- All 'ancestors' of a_c must be proceeding activities. This is a postcondition rather than a precondition as we might not want to start all 'higher level' activities explicitly - the starting of a part implies that the whole has started, and may well be that start.

ActClass3.Start(a)

Pr 22: **ActClass2.Start**(a)
Po 14: $(im\ During^{+})\ \{a\} \subseteq Proceed'$

Preconditions:

- Invoke Start

Postconditions:

- All 'ancestors' of a must be proceeding activities

ActClass3.Suspend(a)

Pr 23: (im During ° Includes) {a} ⊆ {a}

Pr 24: Includes {a} ⊆ Request ∪ Complete

Pr 25: **ActClass2.Suspend**(a)

Preconditions:

- a has no 'spouses' via the During relation. Ie, ie a does not have any offspring that have a parent other than a.
- All activities that are During a are either in Request or Complete - none is in Proceed.
- Inherit the preconditions and postconditions of the Suspend operation.

ActClass3.End(a)

Pr 26. (im Includes) {a} ⊆ Complete

Pr 27: **ActClass2.End**(a)

Preconditions:

- All activities that are Included in a are already complete.
- Inherit the preconditions and postconditions of the End operation

ActClass3.Cancel(a)

Pr 28: (im Includes$^+$) {a} ⊆ Request

Pr 29: **ActClass2.Cancel**(a)

Po 15: (im Includes$^+$) {a} ∩ Activities' = ∅

Preconditions:

- All activities that are 'descendants' of a are in Request. This means that a suspended activity that has completed components cannot be cancelled. An activity that has no completed descendants that has been started and then suspended can be completed. This is an unrealistic situation and does not convey the meaning of the operation: it is a shortcoming of the theory.
- Inherit the preconditions and postconditions of the Cancel operation.

Postconditions:

- None of the descendants of a is now an activity.

ActClass3.OutCreate(a)

Pr 30: **ActClass2.OutRequest**(a)

Preconditions:

- Inherit the preconditions and postconditions of the operation OutRequest. This schema merely changes the name of the operation..

ActClass3.OutEmbed(A_p,a_c)

Pr 31: A_p : Activities\Complete
Pr 32: $A_p \subseteq$ In
Pr 33: **ActClass2.OutRequest**(a_c)
Po 16: $\{a_c\} \times A_p \subseteq$ During'

Preconditions:

- A_p is a set of activities that have not been completed.
- All of A_p is in In
- Inherit the preconditions and postconditions of the operation OutRequest

Postconditions:

- a_c is During all of A_p.

ActClass3.OutProceed(a)

Pr 34: **ActClass2.OutProceed**(a)
Po 17: During$^+$ {a} $\subseteq$ Proceed'

Preconditions:

- Inherit the preconditions and postconditions of the operation OutProceed

Postconditions:

- Any activity that is a descendant of a is in Proceed.

ActClass3.OutComplete(a)

Pr 35: Includes(a) $\subseteq$ Complete
Pr 36: **ActClass2.OutComplete**(a)
Po 18: During$^+$ {a} $\cap$ Request' = $\varnothing$

Preconditions:

- All activities that are Included in a are in Complete
- Inherit the preconditions and postconditions of the OutComplete operation

Postconditions:

- None of the activities that a is Included in are in Request.

PatientClass

T4: Patients, PatPres: Set[P]
I13: PatPres ⊆ Patients
Pats' = ∅

Types:

- Patients and PatPres represent sets of people. Patients is all people who are registered with the organisation. PatPres are all registered patients who are physically present in the organisation. A registered patient is one who is recognised by the directorate - they may or may not be registerd in the official sense.

Invariants:

- PatPres is a subset of Patients.

PatientClass.Register(p)

Pr 37: p: P
Po 19: p: Patients'\PatPres'

Preconditions:

- p is a person (ie, an element of the carrier set P).

Postconditions:

- p is a patient who is not physically present.

PatientClass.Deregister(p)

Pr 38: p: Patients
Po 20: p ∉ Patients'

Preconditions:

- p is a registered patient

Postconditions:

- p is no longer a patient.

NB This operation would be used to remove a patient that had no medical contact with the clinic. It would probably never be used.

PatientClass.Arrive(p)

Pr 39: p: Patients\PatPres
Po 21: p: PatPres'

Preconditions:

- p is a patient who is not present

Postconditions:

- p is present in the organisation.

PatientClass.Depart(p)

Pr 40: p: PatPres
Po 22: $p \notin PatPres'$

Preconditions:

- p is present

Postconditions:

- p is not present in the organisation.

◇◇◇

Preconditions:

- p is a person - not necessarily a registered patient
- All the activities to be before the new one must have p as their subject.
- Invoke InCreate

Postconditions:

- p is the subject of the new activity
- p is a registered patient.

APClass1.Embed(A_p,A_b,a_c)

Pr 44: $\forall a1,a2: A_p \cup A_b \bullet ActSubject(a1) = ActSubject(a2)$

Pr 45: **ActClass3.Embed**(A_p,A_b,a_c)

Po 25: $a_c \in Dom(ActSubject')$

Po 26: $ActSubject'(a_c) \in (im\ ActSubject)\ A_p$

Preconditions:

- All activities in A_b and A_p must have the same subject.
- Inherit the preconditions and postconditions of the InEmbed operation

Postconditions:

- a_c has a subject
- the subject of a_c is the same as the subject of an activity in A_p.

APClass1.SuddenStart(A_p,a_c)

Pr 46: **ActClass3.SuddenStart**(A_p,a_c)

Po 27: $a_c \in Dom(ActSubject')$

Po 28: $ActSubject'(a_c) \in (im\ ActSubject)\ A_p$

Preconditions:

- Inherit the preconditions and postconditions of the SuddenStart operation

Postconditions:

- a_c has a subject
- the subject of a_c is the same as the subject of an activity in A_p.

APClass1.Start(a)

APClass1.Suspend(a)

Pr 47: **ActClass3.Suspend**(a)

Po 29: $a \notin ActAtt'$

Preconditions:

- Inherit the preconditions and postconditions of the Suspend operation

Postconditions:

- a is not an attended activity

APClass1.End(a)

APClass1.Cancel(a)

APClass1.OutCreate(p,a)

Pr 48: p: Patients
Pr 49: **ActClass3.OutCreate**(a)
Po 30: (a,p) ∈ ActSubject'

Preconditions:

- p is a registered patient. The patient must be registered as we are not interested in patients that have no contact with our organisation.
- Inherit the preconditions and postconditions of the OutCreate operation.

Postconditions:

- p is the subject of a.

APClass1.OutEmbed(A_p,a_c)

Pr 50: $\forall a1,a2: A_p \bullet ActSubject(a1) = ActSubject(a2)$
Pr 51: **ActClass3.OutEmbed**(Ap,ac)
Po 31: $a_c \in Dom(ActSubject')$
Po 32: $ActSubject'(a_c) \in (im\ ActSubject)\ A_p$

Preconditions:

- All activities in A_p must have the same subject.
- Inherit the preconditions and postconditions of the operation OutEmbed

Postconditions:

- a_c has a subject
- the subject of a_c is the same as the subject of an activity in A_p.

APClass1.OutProceed(a)

APClass1.OutComplete(a)

APClass1.PatReg(p)

APClass1.PatDereg(p)

APClass1.PatArrive(p)

APClass1.PatDepart(p)

APClass1.PatJoin(a,p)

Pr 52: p: PatPres
Pr 53: a: In\Complete
Pr 54: p = ActSubject(a)
Po 33: a ∈ ActAtt'

Preconditions:

- p is a patient present in the organisation
- a is not a complete activity
- p is the subject of a

Postconditions:

- a is an attended activity.

APClass1.PatLeave(p)

Pr 55: p: (im ActSubject) ActAtt
Po 34: (im ActSubject^{-1}) {p} ∩ ActAtt = ∅

Preconditions:

- p is a patient that is the subject of an attended activity

Postconditions:

- There are no activities that have p as their subject that are attended activities.

TypeClass1

T 7: Types, Unplanned, Access, PatReq, HomeTypes: DT ⇸ Org
T 8: Home: Set[Org]
I 16: Home ⊆ Cod(Types)
I 17: PatReq, Access, HomeTypes, Unplanned ⊆ Types
I 18: HomeTypes = Types ⩥ Home
I 19: Access ∩ Unplanned = ∅
I 20: PatReq ⊆ HomeTypes
Home' = Types' = ∅

Types:

- Types is a pair with medical descriptions as the domain and organisations as the codomain. The description is the medical name of a clinically relevant class of activities - for example 'Blood Test', 'Doctor Consultation', or 'Diabetic Care'. Because the description is paired with organisation in the state component Types, a Doctor Consultation conducted within the Diabetes Directorate can be considered to be a different type of activity from a Doctor Consultation conducted within the

Obstetrics and Gynaecology Directorate. If this were the case then the pairs (Doctor Consultation, Diabetes Directorate) and (Doctor Consultation, Obstetrics and Gynaecology Directorate) would be elements of Types. Unplanned, Access, PatReq and HomeTypes are similarly defined. Unplanned is the set of Types, activities of which can be started without previously having been requested (ie via the SuddenStart operation). Access is the set of Types, activities of which can be created without being embedded in another activity - these are the types that can be 'accessed' from outside the organisation. PatReq is the set of Types, activities of which need a patient to be present before they can start. HomeTypes is the set of Types, activities of which can be considered to be the responsibility of the 'home' organisation.

- Home is the set of organisations that represent the administrative groupings that we are directly interested in. This will generally be a singleton set (for example, {Diabetes and Endocrine Directorate}).

Invariants:

- The set Types must have any Home organisation in the set Home in its codomain.
- PatReq, Access, HomeTypes and Unplanned are all subsets of Types.
- HomeTypes is the largest subset of Types that has its codomain equal to the set Home.
- No Types in Access can also be in Unplanned.
- Types in PatReq must also be in HomeTypes

In all the specialisation classes (TypeClass1, TypeClass2, TypeClass3, TypeClass4, TypeClass5, TypeClass6, TypeClass7, Clin1, Scheduler, and even Clock) there is an implied operation that is not specified. This might be called .OrgDef (for Organisation Define), or .Specialise, or some similar name. This operation is invoked directly after the initialisation of the object of the class (or any class that inherits properties of a specialisation class - ie all classes 'after' ActClass3): no operation can be invoked before .Specialise. The operation creates values for all the specialisation state components (those declared in the specialisation classes) effectively 'setting up' the model so that it can represent a real domain. This operation clearly should not be interpreted into the world as it does not represent anything real. However, equally clearly, it is necessary if a model of the theory is to be non-vacuous. Although this operation is not specified, it would be easy to do so - one would merely provide arguments to an operation that would become the specialisation state components after invocation, and the preconditions would thus mirror the invariants, only existing over the arguments as opposed to the state components.

ATClass1

APClass1, TypeClass1

T9: ActType: Activities $\rightarrow$ Types

I21: (im ActType) In $\subseteq$ HomeType

I22: (im ActType) Out $\cap$ HomeType = $\varnothing$

I23: ActType $\circ$ Includes^{+} $\circ$ ActType^{-1} $\cap$ id[Types] = $\varnothing$

I24: #(ActType $\circ$ After) = #After

I25: #Cod(Access $\lhd$ (ActSubject $\circ$ ActType^{-1})) = #Cod(ActSubject)

I26: #ActType $\circ$ (Includes $\rhd$ Proceed) = #(Includes $\rhd$ Proceed)

I27: #ActType $\circ$ (Includes $\rhd$ Request) = #(Includes $\rhd$ Request)

I28: #During = #(ActType $\circ$ During)

- Inherit the type declarations and invariants of the classes APClass1 and TypeClass1

Types:

- ActType is a function which returns the type of a particular activity. Each activity must have exactly one type. Furthermore we will see that once a type has been assigned to ab activity it cannot be changed.

Invariants:

- All internal activities must be of a type in HomeType
- No external activities can be of a type in HomeType
- The relation Includes, projected onto Types via ActType is a DAG. In other words, no activity can have an ancestor that is the same type as it is.
- An activity can be after at most one other activity of any given type.
- Any patient that is associated with an activity via ActSubject must be associated with at least one activity of a type in Access.
- For any patient there must be only one activity of any given type proceeding at any time within the same parent activity.
- For any patient there must be only one activity of any given type requested at any time within the same parent activity.
- No two parents of a given activity can be of the same type.
- We do not say that Unplanned activities cannot be requests a they might be suspended.

ATClass1.Create(A_b,p_n,t_n,a_n)

Pr 56: t_n : Access ∩ HomeTypes
Pr 57: #(im ActType) A_b = #A_b
Pr 58: **APClass1.Create**(A_b,p_n,a_n)
Po 35: (a_n,t_n) ∈ ActType'

Preconditions

- t_n is in Access and HomeTypes
- All A_b are of different types
- Inherit the preconditions and postconditions of the InCreate operation

Postconditions

- a_n is now of type t_n as specified by the relation ActType

ATClass1.Embed(A_p,A_b,t_c,a_c)

Pr 59: t_c : HomeTypes

Pr 60: #(im ActType) A_b = #A_b

Pr 61: #(im ActType) A_p = #A_p

Pr 62: $t_c \notin$ (im ActType) (im During*) A_p

Pr 63. $t_c \notin$ (im ActType) ((im Includes) A_p) ▷ Request)

Pr 64: **APClass1.Embed(A_p,A_b,a_c)**

Po 36: $(a_c,t_c) \in$ ActType'

Preconditions:

- t_c is in HomeTypes
- All A_b are of different types
- All A_p are of different types
- None A_p or any of the ancestors of any activity in A_p is of type tc.
- None of the existing children of any of A_p that are in Request are of type t_c.
- Inherit the preconditions and postconditions of the InEmbed operation

Postconditions:

- a_c is now of type t_c as specified by the relation ActType.

ATClass1.SuddenStart(A_p,t_c,a_c)

Pr 65: t_c : HomeTypes ∩ Unplanned

Pr 66: #(im ActType) A_p = #A_p

Pr 67: $t_c \notin$ (im ActType) (im During*) A_p

Pr 68: tc $\notin$ (im ActType) ((im Includes) A_p) ▷ Proceed)

Pr 69: **APClass1.SuddenStart(A_p,a_c)**

Po 37: $(a_c,t_c) \in$ ActType'

Preconditions:

- t_c must be in HomeTypes and Unplanned.
- All A_p are of different types
- None A_p or any of the ancestors of any activity in A_p is of type t_c.
- None of the existing children of any of A_p that are in Proceed are of type t_c.
- Inherit the preconditions and postconditions of the SuddenStart operation

Postconditions:

- a_c is now of type t_c as specified by the relation ActType.

ATClass1.Start(a)

Pr 70. ActType(a) ∈ PatReq ⇒ a ∈ ActAtt

Pr 71: ActType(a) ∉ (im ActType) (((im Includes ° During) {a}) ∩ Proceed)

Pr 72: **APClass1.Start**(a)

Preconditions:

- If a is of a type that needs patients to start then a patient must be present at its start.
- There can be no activities of the same type as a that have the same parent in Proceed
- Inherit the preconditions and postconditions of the Start operation

ATClass1.Suspend(a)

ATClass1.End(a)

ATClass1.Cancel(a)

ATClass1.OutCreate(p,t,a)

Pr 73: t: Access\HomeTypes

Pr 74: **APClass1.OutCreate**(a)

Po 38: (a,t) ∈ ActType'

Preconditions:

- t is in Access but not HomeTypes
- Inherit the preconditions and postconditions of the OutCreate operation

Postconditions:

- a is now of type t as specified by the relation ActType.

ATClass1.OutEmbed(A_p,t_c,a_c)

Pr 75: t_c : Types\HomeTypes

Pr 76: **APClass1.OutEmbed**(A_p,a_c)

Po 39: (a_c,t_c) ∈ ActType'

Preconditions:

- t is a type that is not in HomeTypes
- Inherit the preconditions and postconditions of the OutEmbed operation

Postconditions:

- ac is now of type tc as specified by the relation ActType.

ATClass1.OutProceed(a)

ATClass1.OutComplete(a)

ATClass1.PatReg(p)

ATClass1.PatDereg(p)

ATClass1.PatArrive(p)

ATClass1.PatDepart(p)

ATClass1.PatJoin(a,p)

ATClass1.PatLeave(p)

◇◇◇

ATClass2

ATClass1, TypeClass2

I 32: $\forall a$: Activities • $\exists tg$: TGroupers • (ActType $\circ$ ({a} $\lhd$ During) $\circ$ ActType^{-1} = $\varnothing$) $\vee$
ActType $\circ$ ({a} $\lhd$ During) $\circ$ ActType^{-1} = {ActType(a)} $\lhd$ TypeGuide(tg)

- Inherit the type declarations and invariants of the classes ATClass1 and TypeClass2

Invariants:

- For any activity, there exists an element of the set TGroupers such that the 'parents' of the activity are of types that are in the codomain of the graph that is returned when the member of TGrouper is supplied to TypeGuide. Furthermore, there must be an activity with the appropriate type for each element in the codomain of that graph.

ATClass2.Create(A_b,p_n,t_n,a_n)

ATClass2.Embed(A_p,A_b,t_c,a_c)

Pr 77: $\exists tg$: TGroupers • (im ActType) A_p = (im TypeGuide(tg)) {t_c}

Pr 78: **ATClass1.Embed**(A_p,A_b,t_c,a_c)

Preconditions:

- There is some member of the set TGroupers such that the types of the would be parents of the new activity are identical to the types in the codomain of the graph returned when the element of TGroupers is supplied to TypeGuide.

- Inherit the preconditions and postconditions of the OutEmbed operation

ATClass2.SuddenStart(A_p,t_c,a_c)

Pr 79: $\exists m$: TGroupers • (im ActType) Ap = (im TypeGuide(m)) {t_c}

Pr 80: **ATClass2.SuddenStart**(A_p,t_c,a_c)

Preconditions:

- There is some member of the set TGroupers such that the types of the would be parents of the new activity are identical to the types in the codomain of the graph returned when the element of TGroupers is supplied to TypeGuide.

- Inherit the preconditions and postconditions of the SuddenStart operation

ATClass2.Start(a)

ATClass2.Suspend(a)

ATClass2.End(a)

ATClass2.Cancel(a)

ATClass2.OutCreate(p,t,a)

ATClass2.OutEmbed(A_p,t_c,a_c)

Pr 81: $\exists m$: TGroupers • (im ActType) A_p = (im TypeGuide(m))$^{-1}$ $\{t_c\}$

Pr 82: **ATClass1.OutEmbed(A_p,t_c,a_c)**

Preconditions:

- There is some member of the set TGroupers such that the types of the would be parents of the new activity are identical to the types in the codomain of the graph returned when the element of TGroupers is supplied to TypeGuide.
- Inherit the preconditions and postconditions of the OutEmbed operation

ATClass2.OutProceed(a)

ATClass2.OutComplete(a)

ATClass2.PatReg(p)

ATClass2.PatDereg(p)

ATClass2.PatArrive(p)

ATClass2.PatDepart(p)

ATClass2.PatJoin(a,p)

ATClass2.PatLeave(p)

◇◇◇

Types:

- RunType is a total function from types of activity that are pertinent to the home organisation to types of clinician. RunType records which types of activities can be started by which types of clinician.

Invariants:

- All unplanned activity types must share a running clinician type with all its parents. This invariant is necessary to ensure that the SuddenStart operation is always possible - whereas with 'planned' activities, we can insist on the parents of those activities having already started, this is not appropriate for unplanned activities which must be able to start, and to start all parents if this is necessary.

◇◇◇

ATClass3.Start(a)

Pr 86: a ∈ Dom(ActRun)
Pr 87: **ATClass2.Start**(a)

Preconditions:

- a is a member of the domain of ActRun.
- Inherit the preconditions and postconditions of the Start operation

ATClass3.Suspend(a)

Pr 88: a ∈ Dom(ActRun)
Pr 89: **ATClass2.Suspend**(a)
Po 41: a ∉ Dom(ActRun')

Preconditions:

- a is a member of the domain of ActRun.
- Inherit the preconditions and postconditions of the Suspend operation

Postconditions:

- a is no longer a member of the domain of ActRun. This reflects the meaning of the Suspend operation which is supposed to represent the withdrawal of the clinician (and patient) from the process of the activity, meaning that both the clinician and the patient are free to become engaged in theprocess of another activity.

ATClass3.End(a)

Pr 90: a ∈ Dom(ActRun)
Pr 91: **ATClass2.End**(a)

Preconditions:

- a is a member of the domain of ActRun.
- Inherit the preconditions and postconditions of the End operation

NB We don't need the postcondition of the Suspend activity as our insistence that the domain of ActRun is a subset of incomplete activities means that a is automatically removed from the domain od ActRun after the operation.

ATClass3.Associate(a,hcp)

Pr 92: a: In\(Complete ∪ Dom(ActRun))
Pr 93: hcp: HCP
Pr 94: (ActType(a),ProfType(hcp)) ∈ RunType
Po 42: (a,hcp) ∈ ActRun'

Preconditions:

- a is an internal activity that is not complete and is not in the domain of ActRun

- hcp is an HCP
- The pair consisting of the activity type of a and the HCP type of hcp is an element of RunType.

Postconditions:

- hcp is now associated with a via the function ActRun.

ATClass3.Disassociate(hcp)

Pr 95: hcp: Cod(ActRun)
Pr 96.(im $ActRun^{-1}$) {hcp} $\subseteq$ Request
Po 43: hcp $\notin$ Cod(ActRun)

Preconditions:

- hcp is an element of HCP that is associated with an activity via the relation ActRun.
- All activities that hcp is currently associated with are in the set Request (an HCP cannot disassociate from a proceeding activity).

Postconditions:

- hcp is no longer associated with any activities via ActRun. It doesn't make sense for a clinician to disassociate from fewer than all currently associated activities. The idea behind being associated with several activities was in case one is nested in another. If the clinician leaves one, it will be because he or she has to attend to something else - he or she will then leave them all. The clinician can only disassociate from requests - otherwise he or she must end, cancel or suspend them, which has the same effect.

ATClass3.OutCreate(p,t,a)

ATClass3.OutEmbed(A_p,t_c,ac)

ATClass1.OutProceed(a)

ATClass1.OutComplete(a)

ATClass3.PatReg(p)

ATClass3.PatDereg(p)

ATClass3.PatArrive(p)

ATClass3.PatDepart(p)

ATClass3.PatJoin(a,p)

ATClass3.PatLeave(p)

TypeClass4

TypeClass3

T 16: EmbedType: Pr ⇸ (TGroupers ⇸ (Types ⇿ Types))

T 17: OutRefType: Pr ⇿ Types

I 35: Cod(OutRefType) ⊆ Access

I 36: Cod(EmbedType) ⊆ TypeGuide

I 37: ∀tc: Pr • ∀tg: Dom(EmbedType(tc)) •
Cod(EmbedType(tc)(tg)) = Cod(TypeGuide(tg)) ∧
Dom(EmbedType(tc)(tg)) ⊆ Dom(TypeGuide(tg)) ∧
(Cod(EmbedType(tc)(tg)) ⊆ Cod(RunType ▷ {tc})) ⇒ Cod({tg} ◁ TypeGuide) = EmbedType(tc)(tg)) ∧
(~(Cod(EmbedType(tc)(tg)) ⊆ Cod(RunType ▷ {tc}))) ⇒ Dom(EmbedType(tc)(tg)) ∩ Unplanned = ∅)

- Inherit the type declarations and invariants of the class TypeClass3

Types:

- EmbedType is a function from types of HCP to another relation which is in its turn a function from the set TGroupers to a graph over types. This structure records which types of activity a particular type of clinician can embed in which other types of activity. It does this by effectively providing each type of HCP with a subset of TypeGuide (in fact the situation is slightly more complex than this as we shall see).
- OutRefType is a relation which records which types of activity that are external to the home organisation can be created by which type of clinician.

Invariants:

- The types in the codomain of OutRefType are all in the set Access.
- The codomain of EmbedType is a subset of TypeGuide
- This invariant can best be explained in a number of parts. Note that the third part of the invariant means that the clinican 'responsible' for the running of an activity will be able to embed (allowable - via TypeGuide) component activities within it. The complexity comes from having to consider the possibility of multiple parents for activities. For all types of HCP (which we shall call tc) and all elements of TGrouper that are in the domain of the function returned when tc is supplied to EmbedType:

  A. if a type of clinician can embed a type of activity in another that is a permissable parent via TypeGuide, then that type of clinician can embed that type of activity in all permissable parents; and

  B. the activity types that can be embedded in others by tc that are in the graph referenced by tg must be in the domain of tg; and

  C. if an HCP of type tc can run all the activity types that are possible parents of types in the domain of the graph within EmbedType referenced by tg, then that type of clinician can embed all allowable types in the types represented by that element; and

  D. if an HCP of type tc cannot run all the activity types that are possible parents of types in the domain of the graph within EmbedType referenced by tg, then none of the embeddable types can be Unplanned.

ATClass4

ATClass3, TypeClass4

I 38: ∀a1,a2: Activities\Complete • (ActType(a1) = ActType(a2) ∧ ActSubject(a1) = ActSubject(a2) ∧ ActType(a1) ∈ Cod(OutRefType) ⇒ a1 = a2

- Inherit the type declarations and invariants of the classes ATClass3 and TypeClass4

Invariants:

- If there are two incomplete activities that are of the same type, concern the same patient, and one of whose type is in the codomain of OutRefType then those two activities are identical.

ATClass4.Create(A_b,p_n,t_n,a_n)

ATClass4.Embed(A_p,A_b,t_c,hcp,a_c)

Pr 97: hcp: HCP

Pr 98: ∃tg: TGroupers • (im ActType) A_p = (im EmbedType(ProfType(hcp))(tg)) $\{t_c\}$

Pr 99: **ATClass3.Embed**(A_p,A_b,t_c,a_c)

Preconditions:

- hcp is a member of the set HCP
- There is a member of TGroupers, tg, such that all the would be parents of the new activity are of the types specified for the HCP type of hcp, the member of TGroupers tg, and the type of the new activity, in the structure EmbedType.
- Inherit the preconditions and postconditions of the Embed operation

ATClass4.SuddenStart(A_p,t_c,hcp,a_c)

Pr 100: ∃tg: TGroupers • (im ActType) A_p = (im EmbedType(ProfType(hcp))(tg)) $\{t_c\}$

Pr 101: **ATClass3.SuddenStart**(A_p,t_c,hcp,a_c)

Preconditions:

- There is a member of TGroupers, tg, such that all the would be parents of the new activity are of the types specified for the HCP type of hcp, the member of TGroupers tg, and the type of the new activity, in the structure EmbedType.
- Inherit the preconditions and postconditions of the SuddenStart operation

ATClass4.Start(a)

ATClass4.Suspend(a)

ATClass4.End(a)

ATClass4.Cancel(a)

ATClass4.Associate(a,hcp)

ATClass4.Disassociate(hcp)

ATClass4.OutCreate(p_n,t_n,hcp,a_n)

Pr 102: hcp: HCP

Pr 103: $((\text{im ActType}^{-1})\{t_n\} \cap (\text{im ActSubject}^{-1})\{p_n\})\backslash\text{Complete} = \varnothing$

Pr 104: $(\text{ProfType(hcp)},t_n) \in \text{OutRefType}$

Pr 105: **ATClass3. OutCreate**(p,t,a)

Preconditions:

- hcp is a member of HCP
- There are no incomplete activities of the type requested concerning the patient in question.
- An HCP of the type of hcp can create an activity of type t_n as specified by OutRefType.
- Inherit the preconditions and postconditions of the OutCreate operation

ATClass4.OutEmbed(A_p,t_c,hcp,a_c)

Pr 106: hcp: HCP

Pr 107: $\exists$tg: TGroupers • $(\text{im ActType})\ A_p = (\text{im EmbedType}(\text{ProfType(hcp)})(\text{tg}))\ \{t_c\}$

Pr 108: **ATClass3.OutEmbed**(A_p,t_c,a_c)

Preconditions:

- hcp is a member of HCP
- There is a member of TGroupers, tg, such that all the would be parents of the new activity are of the types specified for the HCP type of hcp, the member of TGroupers tg, and the type of the new activity, in the structure EmbedType.
- Inherit the preconditions and postconditions of the OutEmbed operation

ATClass4.OutProceed(a)

ATClass4.OutComplete(a)

ATClass4.PatReg(p)

ATClass4.PatDereg(p)

ATClass4.PatArrive(p)

ATClass4.PatDepart(p)

ATClass4.PatJoin(a,p)

ATClass4.PatLeave(p)

Clock

T 18: Now: $\mathbb{T}$

T 19: Earlier, Later: $\mathbb{T} \leftrightarrow \mathbb{T}$

T 20: Next, Previous: $\mathbb{T} \rightarrow \mathbb{T}$

I 39: $Earlier = Earlier^{+}$

I 40: $Earlier \cap id[\mathbb{T}] = \varnothing$

I 41: $Later = Earlier^{-1}$

I 42: $Later \cap Earlier = \varnothing$

I 43: $Later \cup Earlier \cup id[\mathbb{T}] = \mathbb{T} \times \mathbb{T}$

I 44: $Next = Previous^{-1}$

I 45: $Next \subseteq Later$

I 46: $\forall(\tau1,\tau2): Next \bullet \sim\exists\, \tau3: \mathbb{T} \bullet (\tau1,\tau3) \in Earlier \wedge (\tau2,\tau3) \in Later$

Now' = 12:00am, 1/1/94

Earlier = Later = Next = Previous

- Inherit the type declarations and invariants of the class Constants

Types:

- Now is a member of the carrier set $\mathbb{T}$. $\mathbb{T}$ is the (infinite) set of times. This class defines standard relations over times and is thus not very informative. In fact, it might have been better to map times onto natural numbers and then had a model for the relations specified above. The direct definition above is clear if long winded.
- Earlier and Later are graphs over times. These are similar to the relations < and > in natural numbers. Thus the set (im After) {t} represents all times after t.
- Next and Previous are functions over times. They are similar to the successor function over natural numbers. Thus Next(t) is the time after t.

Invariants:

- Earlier is transitive
- Earlier is a DAG
- Later is the inverse of Earlier
- The intersection of Later and Earlier is null. This means that Later and Earlier are not reflexive.
- When joined with the identity of $\mathbb{T}$, the union of Later and Earlier is the cartesian product of $\mathbb{T}$ with itself (ie $\mathbb{T}^2$). This means that any pair of times is either in Earlier, Later, or the identity function.
- Next is the inverse of Previous.
- Next is a subset of Later
- For any pair in Next, there is no time earlier than the first and later then the second.

Clock.Tick()

Po 44: Now' = Next(Now)

Postconditions:

- Now is the next time after the previous Now.

NB Although this class has a specified operation - Tick - it is really a specialisation class as subsequent classes assume that Earlier, Later, Next and Previous are set up.

◇◇◇

Scheduler.AddSlot(τ1,τ2,c,s)

Pr 109: s: S; τ1,τ2: T;c: C

Pr 110: s ∈ Slots ⇒ SlotStart(s) = τ1 ∧ SlotEnd(s) = τ2 ∧ SlotClist(s) = c

Pr 111: (τ1,τ2) ∈ Earlier

Pr 112: ∀s2: Im SlotClinic^{-1} {c} • ((τ1,SlotStart(s2)) ∈ Later ⇒ (τ1,SlotEnd(s2)) ∈ Later) ∧
((τ2,SlotEnd(s2)) ∈ Earlier ⇒ (τ2,SlotStart(s2)) ∈ Earlier)

Pr 113: (τ1,Now) ∈ Later

Po 45: s ∈ Slots'

Po 46: SlotStart'(s) = τ1; SlotEnd'(s) = τ2

Po 47: SlotClist'(s) = c

Preconditions:

- s is a potential slot (although it might already exist)
- τ1 and τ2 are times
- c is a clinic list
- If s is already a slot, then τ1 must be its start, τ2 it end, and c its clinic list.
- For any other slot in the clinic list c, if it starts before τ1, it finishes before τ2, and if it finishes after τ2, it starts after τ2.
- The start of the slot is in the future.

Postconditions:

- s is now in Slots
- SlotStart is τ1
- SlotEnd is τ2
- SlotClist is c.

Scheduler.DelSlot(s)

Pr 114: s: Slots

Po 48: s ∉ Slots'

Preconditions:

- s is a slot

Postconditions:

- s is not a slot

Scheduler.Tick()

TypeClass5

TypeClass4

T24: Bookable: Set[Types]

I50: Bookable ∩ Unplanned = ∅

I51: Bookable ∩ Access = ∅

- Inherit the type declarations and invariants of the class TypeClass4

Types:

- Bookable is a set of Types. Bookable is all those types of activity that *can* be booked. This does not mean that they *must* be booked however.

Invariants:

- No Bookable activity type can also be in Unplanned
- No Bookable activity type can be directly accessed from outside the clinic (?).

◇◇◇

- All activities that are during others have their request times recorded.
- All activities that have slots associated with them are of a type in Bookable.
- All completed internal activities started as many times as they ended.

NB although an activity cannot be associated with more than one slot, more than one activity can be associated with a slot. Thus patient several patient education sessions may be associated with the same slot: they all start and end at the same time in the same room, run by the same professionals.

ATClass5.Create(A_b,p_n,t_n,a_n)

Pr 115: **ATClass4.Create**(A_b,p_n,t_n,a_n)
Po 49: (a_n,now) ∈ ActReq'

Preconditions:

- Inherit the preconditions and postconditions of the Create operation

Postconditions:

- The new activity has the time now recorded as its time of request.

ATClass5.Embed(A_p,A_b,t_c,hcp,a_c)

Pr 116: **ATClass4.Embed**(A_p,A_b,t_c,hcp,a_c)
Po 50: (a_n,now) ∈ ActReq'

Preconditions:

- Inherit the preconditions and postconditions of the Embed operation

Postconditions:

- The new activity has the time now recorded as its time of request.

ATClass5.Book(A_p,A_b,t_c,hcp,c,τ_b,τ_e,a_c,s)

Pr 117: t_c ∈ Bookable
Pr 118: **ATClass5.Embed**(A_p,A_b,t_c,hcp,a_c)
Pr 119: **ATClass5.AddSlot**(c,τ_b,τ_e,s)
Po 51: (a_c,now) ∈ ActReq'
Po 52: (a_c,s) ∈ ActSlot'

Preconditions:

- The activity to be created is of a type in Bookable
- Inherit the preconditions and postconditions of the Embed operation
- Inherit the preconditions and postconditions of the AddSlot operation

Postconditions:

- The new activity has the time now recorded as its time of request.
- The new activity has the (possibly newly created slot) s allocated to it via ActSlot

ATClass5.Book(c,τ_b,τ_e,a_c,s)

Pr 120: a_c : Request\Dom(ActSlot)
Pr 121: ActType(a_c) ∈ Bookable
Pr 122: **ATClass4.AddSlot**(c,τ_b,τ_e,s)
Po 53: (a_c,s) ∈ ActSlot'

Preconditions:

- The activity to be assigned a slot (ie, the activity to be booked) must be a request that has not already been assigned to a slot
- The type of the activity being booked must be in Bookable
- Inherit the preconditions and postconditions of the AddSlot operation

Postconditions:

- The new activity has the (possibly newly created slot) s allocated to it via ActSlot

ATClass5.Unbook(a)

Pr 123: a: Request ∩ Dom(ActSlot)
Po 54: a ∉ Dom(ActSlot')

Preconditions:

- The activity to be unbooked must be in Request and already be assigned to a slot.

Postconditions:

- The activity is no longer assigned to a slot.

ATClass5.SuddenStart(A_p,t_c,hcp,a_c)

Pr 124: **ATClass4.SuddenStart**(A_p,t_c,hcp,a_c)
Po 55: (a,now) ∈ ActStart' ∩ ActReq'

Preconditions:

- Inherit the preconditions and postconditions of the SuddentStart operation

Postconditions:

- The new activity has the time now recorded as its time of request and its time of start

ATClass5.Start(a)

Pr 125: **ATClass4.Start**(a)
Po 56: (a,now) ∈ ActStart'

Preconditions:

- Inherit the preconditions and postconditions of the Start operation

Postconditions:

- The started activity has the time now recorded as its time of start

ATClass5.Suspend(a)

Pr 126: **ATClass4.Suspend**(a)
Po 57: (a,now) ∈ ActEnd'

Preconditions:

- Inherit the preconditions and postconditions of the Suspend operation

Postconditions:

- The suspended activity has the time now recorded as its time of completion

ATClass5.End(a)

Pr 127: **ATClass4.End**(a)
Po 58: (a,now) ∈ ActEnd'

Preconditions:

- Inherit the preconditions and postconditions of the End operation

Postconditions:

- The completed activity has the time now recorded as its time of completion

ATClass5.Cancel(a)

Pr 128: (im Includes*) {a} ∩ Dom(ActStart) = ∅
Pr 129: **ATClass4.Cancel**(a)

Preconditions:

- Neither the activity to be cancelled, nor any of its parents, can have been started.
- Inherit the preconditions and postconditions of the Cancel operation.

ATClass5.Associate(a,hcp)

ATClass5.Disassociate(hcp)

ATClass5.OutCreate(p_n,t_n,hcp,a_n)

Pr 130: **ATClass4.OutCreate**(p_n,t_n,hcp,a_n)
Po 59: (a_n,now) ∈ ActReq'

Preconditions:

- Inherit the preconditions and postconditions of the OutCreate operation

Postconditions:

- The new activity has the time now recorded as its time of request

ATClass5.OutEmbed(A_p,t_c,hcp,a_c)

Pr 131: **ATClass4.OutEmbed**(A_p,t_c,hcp,a_c)
Po 60: (a_c,now) ∈ ActReq'

Preconditions:

- Inherit the preconditions and postconditions of the OutEmbed operation

Postconditions:

- The new activity has the time now recorded as its time of request

ATClass5.OutProceed(a)

Pr 132: **ATClass4.OutProceed**(a)
Po 61: (a,now) ∈ ActStart'

- Inherit the preconditions and postconditions of the OutEmbed operation

Postconditions:

- The started activity has the time now recorded as its time of commencement

ATClass5.OutComplete(a)

Pr 133: **ATClass4.OutComplete**(a)
Po 62: (a,now) ∈ ActEnd'

Preconditions:

- Inherit the preconditions and postconditions of the OutComplete operation

Postconditions:

- The completed activity has the time now recorded as its time of completion

ATClass5.PatReg(p)

ATClass5.PatDereg(p)

ATClass5.PatArrive(p)

ATClass5.PatDepart(p)

ATClass5.PatJoin(a,p)

ATClass5.PatLeave(p)

◇◇◇

TypeClass6

TypeClass5

T28: FollowGuide: Types ⇸ (Types ⇸ Types)

I57: Cod(∪Cod(FollowGuide)) ⊆ Bookable

I58: ∀t1,t2,t3,t4: Types • (t1,t2,t3) ∈ FollowGuide ⇒ ∃m: TGroupers •
{(m,t2,t1), (m,t3,t1)} ⊆ TypeGuide ∧ ({(m,t2,t4),(m,t3,t4)} ∩ TypeGuide ≠ ∅ ⇒ t4 = t1)

I59: ∀m: TGroupers, t1,t2,t3: Types • ∀t4,t5: (im TypeGuide(m)) {t1} •
{(t4,t2,t1), (t5,t3,t1)} ⊆ FollowGuide ⇒ t4 = t5

I60: ∀(t1,t2,t3): FollowGuide • (im RunType) {t1} ∩ (im RunType) {t2} ∩ (im RunType) {t3} ≠ ∅

- Inherit the type declarations and invariants of the class TypeClass5

Types:

- FollowGuide is a function which returns a tree over Types when supplied with a member of Types. The first type is a parent type: the second a child of that parent, and the third the type of followups allowed for activities of the second type with parents of the first type.

Invariants:

- The subset of Types composed of the third element of the all the triples in FollowGuide is a subset of Bookable. In other words, the followup activity must always be of a type in bookable
- There must be an element of TGroupers such that for that element, the first of a triple in FollowGuide must be the only 'parent' type in TypeGuide for both the second type of the triple and the third
- For every graph in the codomain of TypeGuide that is not a tree (ie the child activity is embedded in multiple parents) that has t1 in its domain, only one of the possible parent types has followup components
- There must be some clinician type that can run activities of every type in any given triple in FollowGuide

<>

ATClass6

ATClass5, TypeClass6

T29: Followsup: Activities ⇸ Activities

I61: Followsup ⊆ After

I62: (Cod(Followsup) ◁ During) ∈ Activities ⇸ Activities

I63: ActType ∘ Followsup ∘ ActType^{-1} ⊆ Cod(FollowGuide)

I64: ∀(a1,a2): Followsup • ∃a3: Activities •
{(a1,a3), (a2,a3)} ⊆ During ∧ (ActType(a3),ActType(a1),ActType(a2)) ∈ FollowGuide

I65: ∀a1,a2: Activities\Complete • (ActType(a1) = ActType(a2) ∧ ActSubject(a1) = ActSubject(a2) ∧
ActType(a1) ∈ Dom(FollowGuide)) ⇒ a1 = a2

- Inherit the type declarations and invariants of the classes ATClass5 and TypeClass6

Types:

- Followsup is a tree over activities. The activity on the domain of the function is the followup of the activity in the codomain

Invariants:

- Followsup is a subset of After
- Activities in the domain of Followsup (ie the followup activities) can only have one parent
- The type of the follow up activity and the type of the followed up activity are a pair in the codomain of FollowGuide
- For every pair of activities, a1 and a2, in Followsup, they must both have the same parent and the triple formed by the type of the parent, the type of the followup and the type of the followed up form a triple in FollowGuide
- No more than one activity of a given type that has allowable component types which are followups can be incomplete at the same time for any given patient. This means that there cannot be a requested Dr Care at the same time as a Proceeding Dr Care. These are largely defined by the followups - one Dr Care must finish before another can be requested.

NB In this class not all of the operations change the state of the system. This is because the purpose of this class is to categorise operations and give them 'realistic' names. Thus a familiar operation (eg referral) is specified, and the times when it invokes a previously introduced operation described. Although at other times it may invoke no previously introduced operation, state changes to the system might be introduced at a future date and placed in a postcondition to a refeinement of this operation.

ATClass6.Introduce(p_n,t_n,a_n)

Pr 134: $t_n \in \text{Dom}(\text{FollowGuide}) \Rightarrow ((\text{im ActType}^{-1})\ \{t_n\} \cap (\text{im ActSubject}^{-1})\ \{p_n\})\backslash\text{Complete}) = \varnothing$

Pr 135: **ATClass5.Create**($\varnothing$,p_n,t_n,a_n)

Preconditions:

- If tn is of a type whose 'children' have followups, then there are no incomplete activities of type t_n that are associated with the patient p_n.
- Inherit the preconditions and postconditions of the Create operation

NB This operation introduces a patient to the clinic.

NewTreat: component of activity run by hcp that has component parts then .Generate

ATClass6.NewTreatment(A_p,A_b,t_c,hcp,a_c)

Pr 136: ProfType(hcp) $\in$ (im RunType) $\{t_c\}$

Pr 137: $t_c \in \text{Dom}(\text{Cod}(\text{TypeGuide}))$

Pr 138: $t_c \in \text{Dom}(\text{FollowGuide}) \Rightarrow$
$((\text{im ActType}^{-1})\ \{t_c\} \cap (\text{im ActSubject}^{-1} \circ \text{ActSubject})\ \{A_p\})\backslash\text{Complete}) = \varnothing$

Pr 139: **ATClass5.Embed**(A_p,A_b,t_c,hcp,a_c)

Preconditions:

- hcp must be of a type that can run activities of the type t_c.
- tc is of a type that can have child activities.

- If tn is of a type whose 'children' have followups, then there are no incomplete activities of type t_n that are associated with the patient associated with the parents of the new activity.
- Inherit the preconditions and postconditions of the Embed operation.

NB This operation creates a new treatment for an existing patient. A new treatment is defined as an activity that can have children.

ATClass6. ReferInt(A_p,t_c,hcp,a_c)

Pr 140: A_p : Set[Activities\Complete]; t_c : HomeTypes; hcp: HCP; a_c : A

Pr 141: ProfType(hcp) $\notin$ (im RunType) $\{t_c\}$

Pr 142: $((\text{im ActType}^{-1})\ \{t_c\} \cap (\text{im ActSubject}^{-1} \circ \text{ActSubject})\ \{A_p\}) \backslash \text{Complete}) = \text{Legalt}_c\text{Acts}$

Pr 143: $\text{Legalt}_c\text{Acts} \neq \emptyset \Rightarrow a_c \in \text{Legalt}_c\text{Acts}$

Pr 144: $\text{Legalt}_c\text{Acts} = \emptyset \Rightarrow$ **ATClass5.Embed**(A_p,$\emptyset$,t_c,hcp,a_c)

Preconditions:

- Ap is a set of incomplete activities; tc is a type in HomeTypes; hcp is a member of HCP; ac is a member of the carrier set A.
- hcp is not of a profession type that can 'run' activities of type tc.
- Let the set of incomplete activities that are of type tc and are associated with the same patient as the parents of the new activity be known as $\text{Legalt}_c\text{Acts}$ (All the legal activities that the patient could be referred to)
- If $\text{Legalt}_c\text{Acts}$ is not empty, then the activity the patient is referred to is one of those 'legal' activities.
- If $\text{Legalt}_c\text{Acts}$ is empty, then inherit the preconditions and postconditions of the Embed operation (the referral creates a new activity).

NB This activity creates an internal referral.

ATClass6.ReferExt(p_n,t_n,hcp,a_n)

Pr 145: p_n: Patients; t_n : Types\HomeTypes; hcp: HCP; a_n : A

. 1 Pr 146: $a_n \in \text{Activities}\backslash\text{Complete} \wedge \text{ActType}(a_n) = t_n \wedge \text{ActSubject}(a_n) = p_n \wedge t_n \in \text{Dom}(\text{OutRefType})$

. 2 Pr 147: $(\text{im ActSubject}^{-1})\ \{p_n\} \setminus \text{Complete} \cap (\text{im ActType}^{-1})\ \{t_n\} = \emptyset$

Pr 148: **ATClass5.OutCreate**($\emptyset$,p_n,t_n,hcp,a_n)

Preconditions:

- p_n is a patient; t_n is a type that is not in HomeTypes; hcp is a member of HCP; a_n is in the carrier set A.

Case 1:

Preconditions

- a_n is an incomplete activity, and t_n is the type of a_n, and p_n is the subject of a_n, and t_n is in the domain of OutRefType.

Case 2:

Preconditions

- There are no incomplete activities that have p_n as their subject and are of the type t_n.
- Inherit the preconditions and postconditions of the OutCreate operation.

NB This operation creates an external referral.

ATClass6.Order(A_p,A_b,t_c,hcp,a_c)

Pr 149:(t_c,ProfType(hcp)) ∉ RunType

Pr 150: t_c ∉ Dom(FollowGuide)

Pr 151: t_c ∈ HomeTypes ⇒ **ATClass5.Embed**(A_p,A_b,t_c,hcp,a_c)

Pr 152: t_c ∉ HomeTypes ⇒ **ATClass5.OutEmbed**(A_p,A_b,t_c,hcp,a_c)

Preconditions:

- hcp is not of a type that can run activities of type tc.
- t_c is not in the domain of FollwoGuide (it is not the name for a 'course' of treatment)
- If t_c is in HomeTypes then inherit the preconditions and postconditions of the Embed operation
- If t_c is not in HomeTypes then inherit the preconditions and postconditions of the OutEmbed operation

NB This operation creates an activity that is a component of activity (/ies) run by hcp that is not itself run by hcp and has no followups. This would be invoked to, say, order a test.

ATClass6.Arrange(A_p,A_b,t_c,hcp,a_c)

Pr 153.(t_c,ProfType(hcp)) ∈ RunType

Pr 154: t_c ∉ Dom(Cod(TypeGuide))

Pr 155: **ATClass5.Embed**(A_p,A_b,t_c,hcp,a_c)

Preconditions:

- hcp is of a type that can run activities of the type t_c
- Activities of type t_c may not have children
- Inherit the preconditions and postconditions of the Embed operation

NB This operation creates an activity that is a component of activity (/ies) run by hcp that is also run by hcp and has no possible children. This would be invoked to, say, arrange an Initial Doctor Consultation.

ATClass6.Book(A_p,A_b,t_c,hcp,c,τ_b,τ_e,a_c,s)

ATClass6.Book(c,τ_b,τ_e,a_c,s)

ATClass6.Followup(a_p,a_b,hcp,a_c)

Pr 156: $(a_p,a_b) \in$ Includes

Pr 157: $(\text{ActType}(a_p), t_c, \text{ActType}(a_b)) \in$ FollowGuide

Pr 158: $\{\text{ActType}(a_p), t_c, \text{ActType}(a_b)\} \subseteq \text{Dom}(\text{RunType} \rhd \{\text{ProfType}(hcp)\})$

Pr 159: **ATClass5.Embed**($\{a_p\}$,$\{a_b\}$,FollowGuide(ActType(a_p))(ActType(a_b)),hcp,a_c)

Po 63: $(a_c,a_b) \in$ Followsup'

Preconditions:

- ab is During ap
- The type of the parent activity (a_p) is in the domain of FollowGuide, and t_c is the type that follows the type of a_b when both are components of a_p.
- Each of the types of a_p and a_b, and the type of the new activity (ie t_c) are types that can be run by members of HCP of the same type as hcp.
- Inherit the preconditions and postconditions of the Embed operation

Postconditions:

- ac is now the followup activity of a_b

NB This operation creates followup activities.

ATClass6.SuddenStart(A_p,t_c,hcp,a_c)

ATClass6.Start(a)

ATClass6.Suspend(a)

ATClass6.End(a)

ATClass6.Cancel(a)

ATClass6.Associate(a,hcp)

ATClass6.Disassociate(hcp)

ATClass6.OutProceed(a)

ATClass6.OutComplete(a)

ATClass6.PatReg(p)

ATClass6.PatDereg(p)

ATClass6.PatArrive(p)

ATClass6.PatDepart(p)

ATClass6.PatJoin(a,p)

ATClass6.PatLeave(p)

Information

T30: Records: Set[R]
T31: RecCont: Records → I
Records' = Ø

Types:

- Records is a set of elements taken from a carrier set R. It is the set of identifiers of 'items' of information.
- RecCont is a total function from Records to a carrier set I. The value returned from RecCont when supplied with a member of Records is the 'information', in whatever form it might be, contained in the record identified by the element of Records.

NB The existence of the Information class enables us to talk about the contents of medical records in a totally abstract way, without worrying about what the information is. If we were to expand this class and develop it further, we would probably nbot call it Information as it should reflect our understanding of the patient, not our understanding of the record of the patient's condition (the implemented record system is not interpreted as a record, but as a statement of the patient's state of health). The class is extremely simple at present and only acts as a placeholder for further development.

Information.Add(i,r)

Pr 160: i: I
Pr 161: r: R\Records
Po 64: (r,i) ∈ RecCont'

Preconditions:

- i is a member of the carrier set I.
- r is a member of the carrier set R, but not of Records.

Postconditions:

- i is now the information referred to by r in the function RecCont.

Information.Delete(r)

Pr 162: r: Records
Po 65: r ∉ Records'

Preconditions:

- r is a member of Records.

Postconditions:

- r is not a member of Records (the entry has been removed).

TypeClass7

TypeClass6

T 32: OutNonCont: Set[Types]

I 66: OutNonCont ⊆ Types\(HomeTypes ∪ Access)

Types:

- OutNonCont is a set of Types. OutNonCont is the set of externally run activity types that are known to be completed as soon as information is available for them. An example of this would be the blood test - as soon as we get the results of the test, we know that the test has been completed.

Invariants:

- OutNonCont is a subset of external activity types, and it cannot contain any types in Access.

◇◇◇

ATClass7.ReferInt(a_s,A_p,t_c,hcp,i,r,a_c)

Pr 165: a_s : Proceed

Pr 166: ActSubject(a_s) ∈ (im ActSubject) A_p

Pr 167: (a_s,hcp) ∈ ActRun

Pr 168: ~∃a: (im $ActRun^{-1}$) {hcp} • (a,a_s) ∈ During

Pr 169: **Information.Add**(i,r)

Pr 170: **ATClass6.ReferInt**(a_s,A_p,t_c,hcp,i,r,a_c)

Po 67: (r,a_s) ∈ RecSource'

Preconditions:

- a_s is a proceeding activity (it is the activity from which the internal referral was made - the source of the record)
- The subject of a_s is the same as that of the parent activities.
- The invoker of the operation must be running the source activity.
- The source activity cannot be during another activity that the invoker is running.
- Inherit the preconditions and postconditions of the .Add operation
- Inherit the preconditions and postconditions of the .ReferInt operation

Postconditions:

- a_s is the activity that r is linked to via RecSource.

ATClass7.ReferExt(a_s,p_n,t_n,hcp,i,r,a_n)

Pr 171: a_s : Proceed

Pr 172: ActSubject(a_s) = p_n

Pr 173: (a_s,hcp) ∈ ActRun

Pr 174: ~∃a: (im $ActRun^{-1}$) {hcp} • (a,a_s) ∈ During

Pr 175: **Information.Add**(i,r)

Pr 176: **ATClass6.ReferExt**(a_s,A_p,t_c,hcp,a_c)

Po 68: (r,a_s) ∈ RecSource'

Preconditions:

- a_s is a proceeding activity (it is the activity from which the internal referral was made - the source of the record)
- p_n is the subject of a_s.
- The invoker of the operation must be running the source activity.
- The source activity cannot be during another activity that the invoker is running.
- Inherit the preconditions and postconditions of the .Add operation
- Inherit the preconditions and postconditions of the .ReferExt operation

Postconditions:

- a_s is the activity that r is linked to via RecSource.

ATClass7.Order(A_p,A_b,t_c,hcp,a_c)

ATClass7.Arrange(A_p,A_b,t_c,hcp,a_c)

ATClass7.Book(A_p,A_b,t_c,hcp,c,τ_b,τ_e,a_c,s)

ATClass7.Book(c,τ_b,τ_e,a_c,s)

ATClass7.Followup(a_p,a_b,hcp,a_c)

ATClass7.SuddenStart(A_p,t_c,hcp,a_c)

ATClass7.NoteTake(hcp,a,i,r)

Pr 177: a: Proceed ∩ In; hcp: HCP

Pr 178: (a,hcp) ∈ ActRun

Pr 179: ~∃b: (im ActRun^{-1}) {hcp} • (a,b) ∈ During

Pr 180: **Information.Add**(i,r)

Po 69: (r,a) ∈ RecSource'

Preconditions:

- a is an internal proceeding activity.
- hcp is a member of the set HCP.
- hcp is the health care professional running a.
- a cannot be during another activity that hcp is running
- Inherit the preconditions and postconditions of the .Add operation

Postconditions:

- a is the activity that r is linked to via RecSource.

ATClass7.GetOutData(a,i,r)

Pr 181: a: Out\Complete

Pr 182: ActType(a) ∈ OutNonCont ⇒ **ATClass6.OutComplete**(a)

Pr 183: ActType ∉ OutNonCont ⇒ **ATClass6.OutProceed**(a)

Pr 184: **Information.Add**(i,r)

Preconditions:

- a is an incomplete external activity.
- If the type of a is in OutNonCont then inherit the preconditions and postconditions of the .OutComplete operation (we know that the activity must have finished).
- If the type of a is in OutNonCont then inherit the preconditions and postconditions of the .OutProceed operation (we know that the activity must have started).

- Inherit the preconditions and postconditions of the .Add operation

ATClass7.Start(a)

ATClass7.Suspend(a)

ATClass7.End(a)

ATClass7.Cancel(a)

ATClass7.Associate(a,hcp)

ATClass7.Disassociate(hcp)

ATClass7.OutComplete(a)

ATClass7.PatReg(p)

ATClass7.PatDereg(p)

ATClass7.PatArrive(p)

ATClass7.PatDepart(p)

ATClass7.PatJoin(a,p)

ATClass7.PatLeave(p)

Appendix 3:

A Specialisation of the Domain Theory - The Diabetes & Endocrine Day-Centre

This appendix presents a typical specialisation of the domain theory (Class ATClass7). In it the spceialisation state components of the domain theory are given values. The specialisation is that found to be appropriate for the Diabetes & Endocrine Day-Centre, the organisation this thesis has been mainly concerned with.

The following table records values for the state components *Types* (which is a pair composed of a member of DT and of Org), Unplanned, Access, Bookable, PatReq, and OutNonCont (introduced in TypeClass7)

Descriptive Term (DT)	Organisation (Org)	Unplanned	Access	Bookable	PatReq	OutNonCont
Accident & Emergancy	A&E		Yes			
Antenatal Care	Obs & Gynae		Yes			
Auto Neuro Test	DDC				Yes	
Chiropodist Care	DDC		Yes			
Chiropodist Cons	DDC			Yes	Yes	
Chiropodist Telephone	DDC	Yes				
DECS Care	DDC		Yes			
DECS Cons	DDC			Yes	Yes	
DECS Telephone	DDC	Yes				
Diabetic Care	DDC		Yes			
Dietitian Care	DDC					
Dietitian Cons	DDC			Yes	Yes	
Dietitian Telephone	DDC	Yes				
Dr Care	DDC					
Dr EDM Care	DDC					
Dr GDM Care	DDC					
Dr MARS Care	DDC					
Dr Pop-in	DDC	Yes			Yes	
Dr Telephone	DDC	Yes				
DSN Care	DDC					
DSN Cons	DDC	Yes		Yes	Yes	
DSN Telephone	DDC	Yes				
DSN Edcn Session 1	DDC			Yes	Yes	
DSN Edcn Session 2	DDC			Yes	Yes	
EDM Preg Care	DDC					
Finger Prick Test	DDC					
First Dr Cons	DDC			Yes	Yes	
Followup Dr Cons	DDC			Yes	Yes	
GDM Preg Care	DDC					
GP Care	GP					
Infection Test	Chem Path				Yes	Yes
Lipid Test	Chem Path				Yes	Yes
MARS Care	DDC					
MARS Cons	DDC			Yes	Yes	
Obs Clin Cons	DDC			Yes	Yes	
OGTT	DDC				Yes	
Preg Counsel	DDC			Yes		
Vascular Surgery	Surgeons		Yes			
Venepuncture Test	Chem Path					Yes
Venepuncture Test	Haem					Yes

For clarity's sake, we will only specify the organisation of the type from now on when it is not the DDC.

The following table records values for the state component TypeGuide. This is effectively a triple each member of which is composed of a member from the set TGrouper followed by a member of Types followed by another member of Types

TGrouper	x Types	x Types
tg1	Chiropodist Care	GP Care
tg1	DECS Care	GP Care
tg2	Dr Care	Diabetic Care
tg2	DSN Care	Diabetic Care
tg2	Dietitian Care	Diabetic Care
tg2	Chiropodist Care	Diabetic Care
tg2	DECS Care	Diabetic Care
tg2	EDM Preg Care	Diabetic Care
tg2	GDM Preg Care	Diabetic Care
tg2	MARS Care	Diabetic Care
tg3	First Dr Cons	Dr Care
tg3	Followup Dr Cons	Dr Care
tg3	Dr Telephone	Dr Care
tg3	Finger Prick Test	Dr Care
tg3	Venepuncture Test: Chem Path	Dr Care
tg3	Venepuncture Test: Haem	Dr Care
tg4	DSN Cons	DSN Care
tg4	DSN Telephone	DSN Care
tg4	Finger Prick Test	DSN Care
tg4	DSN Edcn Session 1	DSN Care
tg4	DSN Edcn Session 2	DSN Care
tg5	Dietitian Cons	Dietitian Care
tg5	Dietitian Telephone	Dietitian Care
tg5	Finger Prick Test	Dietitian Care
tg5	Lipid Test: Chem Path	Dietitian Care
tg6	Chiropodist Cons	Chiropodist Care
tg6	Chiropodist Telephone	Chiropodist Care
tg6	Finger Prick Test	Chiropodist Care
tg6	Infection Test: Chem Path	Chiropodist Care
tg7	DECS Cons	DECS Care
tg7	DECS Telephone	DECS Care
tg7	Finger Prick Test	DECS Care
tg8	DSN Cons	DSN Care
tg8	DSN Cons	GP Care
tg9	Dietitian Cons	Dietitian Care
tg9	Dietitian Cons	GP Care
tg10	Chiropodist Cons	Chiropodist Care
tg10	Chiropodist Cons	GP Care
tg11	DECS Cons	DECS Care
tg11	DECS Cons	GP Care
tg12	Dr Pop-in	Chiropody Cons
tg13	Dr EDM Care	EDM Preg Care
tg13	Dr EDM Care	Dr Care
tg13	Dr EDM Care	Antenatal Care: Obs & Gynae
tg14	Obs Clin Cons	Dr EDM Care
tg14	Preg Counsel	Dr EDM Care
tg15	Dr GDM Care	GDM Preg Care
tg15	Dr GDM Care	Dr Care
tg15	Dr GDM Care	Antenatal Care: Obs & Gynae
tg16	Obs Clin Cons	Dr GDM Care
tg16	OGTT	GDM Preg Care
tg17	Dr MARS Care	MARS Care
tg17	Dr MARS Care	Dr Care
tg18	MARS Cons	Dr MARS Care
tg18	Auto Neuro Test	Dr MARS Care

The following table records values for the primitive set ProfType.

ProfType
diabetic doctor
DSN
dietitian
chiropodist
physiologist
staff nurse

The following table records values for the state component RunType which is a function from the set Types to the carrier set Pr.

Types	x Pr
Chiropodist Care	chiropodist
Chiropodist Cons	chiropodist
Chiropodist Telephone	chiropodist
Finger Prick Test	clinic nurse
OGTT	clinic nurse
Diabetic Care	diabetic doctor
Dr Care	diabetic doctor
Dr EDM Care	diabetic doctor
Dr GDM Care	diabetic doctor
Dr MARS Care	diabetic doctor
Dr Pop-in	diabetic doctor
Dr Telephone	diabetic doctor
EDM Preg Care	diabetic doctor
First Dr Cons	diabetic doctor
Followup Dr Cons	diabetic doctor
GDM Preg Care	diabetic doctor
MARS Care	diabetic doctor
MARS Cons	diabetic doctor
Obs Clin Cons	diabetic doctor
Preg Counsel	diabetic doctor
Dietitian Care	dietitian
Dietitian Cons	dietitian
Dietitian Telephone	dietitian
DSN Care	DSN
DSN Cons	DSN
DSN Telephone	DSN
DSN Edcn Session 1	DSN
DSN Edcn Session 2	DSN
Auto Neuro Test	physiologist
DECS Care	physiologist
DECS Cons	physiologist
DECS Telephone	physiologist

The following table records values for the state component EmbedType which is a 4-tuple. Each member of the set is a quartet of elelemts from: Pr, TGrouper, Types, and Types.

Pr	TGrouper	Types	Types
diabetic doctor	m2	Diabetic Care	Dr Care
	m2	Diabetic Care	DSN Care
	m2	Diabetic Care	Dietitian Care
	m2	Diabetic Care	Chiropodist Care
	m2	Diabetic Care	DECS Care
	m2	Diabetic Care	EDM Preg Care
	m2	Diabetic Care	GDM Preg Care
	m2	Diabetic Care	MARS Care
	m3	Dr Care	First Dr Cons
	m3	Dr Care	Followup Dr Cons
	m3	Dr Care	Dr Telephone
	m3	Dr Care	Finger Prick Test
	m3	Dr Care	Venepuncture Test: Chem Path
	m3	Dr Care	Venepuncture Test: Haem
	m4	DSN Care	DSN Cons
	m4	DSN Care	DSN Edcn Session 1
	m4	DSN Care	DSN Edcn Session 2
	m5	Dietitian Care	Dietitian Cons
	m13	EDM Preg Care	Dr EDM Care
	m13	Dr Care	Dr EDM Care
	m13	Antenatal Care: Obs & Gynae	Dr EDM Care
	m14	Dr EDM Care	Obs Clin Cons
	m14	Dr EDM Care	Preg Counsel
	m15	GDM Preg Care	Dr GDM Care
	m15	Dr Care	Dr GDM Care
	m15	Antenatal Care: Obs & Gynae	Dr GDM Care
	m16	Dr GDM Care	Obs Clin Cons
	m16	GDM Preg Care	OGTT
	m17	MARS Care	Dr MARS Care
	m17	Dr Care	Dr MARS Care
	m18	Dr MARS Care	MARS Cons
	m18	Dr MARS Care	Auto Neuro Test
DSN	m2	Diabetic Care	Dr Care
	m2	Diabetic Care	DSN Care
	m2	Diabetic Care	Dietitian Care
	m2	Diabetic Care	Chiropodist Care
	m2	Diabetic Care	DECS Care
	m4	DSN Care	DSN Cons
	m4	DSN Care	DSN Edcn Session 1
	m4	DSN Care	DSN Edcn Session 2
	m5	Dietitian Care	Dietitian Cons
	m8	DSN Care	DSN Cons
	m8	GP Care	DSN Cons
Dietitian	m2	Diabetic Care	Dr Care
	m2	Diabetic Care	DSN Care
	m2	Diabetic Care	Dietitian Care
	m2	Diabetic Care	Chiropodist Care
	m2	Diabetic Care	DECS Care
	m4	DSN Care	DSN Cons
	m5	Dietitian Care	Dietitian Cons
	m5	Dietitian Care	Dietitian Telephone
	m5	Dietitian Care	Finger Prick Test
	m5	Dietitian Care	Lipid Test: Chem Path
	m9	Dietitian Care	Dietitian Cons
	m9	GP Care	Dietitian Cons
Chiropodist	m2	Diabetic Care	Dr Care
	m2	Diabetic Care	DSN Care
	m2	Diabetic Care	Dietitian Care
	m2	Diabetic Care	Chiropodist Care
	m2	Diabetic Care	DECS Care

	m4	DSN Care	DSN Cons
	m4	DSN Care	DSN Edcn Session 1
	m4	DSN Care	DSN Edcn Session 2
	m5	Dietitian Care	Dietitian Cons
	m6	Chiropodist Care	Chiropodist Cons
	m6	Chiropodist Care	Chiropodist Telephone
	m6	Chiropodist Care	Finger Prick Test
	m6	Chiropodist Care	Infection Test: Chem Path
	m7	DECS Care	DECS Cons
	m8	DSN Care	DSN Cons
	m10	Chiropodist Care	Chiropodist Cons
	m10	GP Care	Chiropodist Cons
	m12	Chiropody Cons	Dr Pop-in
physiologist	m2	Diabetic Care	Dr Care
	m2	Diabetic Care	DSN Care
	m2	Diabetic Care	Dietitian Care
	m2	Diabetic Care	Chiropodist Care
	m2	Diabetic Care	DECS Care
	m4	DSN Care	DSN Cons
	m5	Dietitian Care	Dietitian Cons
	m6	Chiropodist Care	Chiropodist Cons
	m7	DECS Care	DECS Cons
	m7	DECS Care	DECS Telephone
	m7	DECS Care	Finger Prick Test
	m11	DECS Care	DECS Cons
	m11	GP Care	DECS Cons

The following table records values for the state components HCP (a simple set) and ProfType (a fuction from HCP to ProfType).

HCP	ProfType
Peter	(Peter, diabetic doctor)
Clara	(Clara, diabetic doctor)
Charles	(Charles, diabetic doctor)
Andrew	(Andrew, diabetic doctor)
David	(David, diabetic doctor)
Sara	(Sara, DSN)
Julia	(Julia, DSN)
Gill Mouth	(Gill, dietitian)
Penny	(Penny, dietitian)
Shirley	(Shirley, physiologist)
Jill Foot	(Jill, chiropodist)
Barbara	(Barbara, staff nurse)
Karen	(Karen, staff nurse)

The following table records values for the state components Follow_guide. This is a triple over Types.

Type	x	Type	x	Type
Dr Care		Followup Dr Cons		First Dr Cons
Dr Care		Followup Dr Cons		Followup Dr Cons
DSN Care		DSN Cons		DSN Cons
Dietitian Care		DSN Cons		DSN Cons
Chiropodist Care		Chiropodist Cons		Chiropodist Cons
DECS Care		DECS Cons		DECS Cons
Dr GDM Care		Obs Clin Cons		Obs Clin Cons
Dr EDM Care		Obs Clin Cons		Obs Clin Cons
Dr MARS Care		MARS Cons		MARS Cons

Appendix 4:

A Theory of the Diabeta CRS and its Interaction with the Domain

Information System Structure

This appendix introduces and describes, in an abstract manner, the structure of a departmental clinical record system similar to that being written for the Diabetes and Endocrine Day Centre (Diabeta IV). It then describes how that system is interpreted into the domain with the help of an interaction theory.

The classes CRSClass1, CRSClass2, CRSClass3, CRSTypeClass, and CRSClass4 describe the CRS. The classes CRSTypeInteraction and CRSInteraction comprise the interaction theory for the system.

CRSClass1

crs-T 1: crs-Proceed, crs-Complete, crs-Visit: Set[crs-V]
crs-I 1: crs-Proceed ∩ crs-Complete = ∅
crs-I 2: crs-Proceed ∪ crs-Complete = crs-Visit
crs-Visit' = ∅

Types:

- crs-Proceed, crs-Complete, and crs-Visit are all sets with elements of the type crs-V.

Invariants:

- crs-Proceed and crs-Complete are disjoint.
- crs-Proceed and crs-Complete together form crs-Visit.

CRSClass1.CreVis(v)

crs-Pr 1: v: crs-V\crs-Visit
crs-Po 1: v ∈ crs-Proceed'

Preconditions:

- v is a member of crs-V that has not been transferred to crs-Visit.

Postconditions:

- v is now a member of crs-Proceed.

CRSClass1.FinVis(v)

crs-Pr 2: v: crs-Proceed
crs-Po 2: v ∈ crs-Complete'

Preconditions:

- v is a member of crs-Proceed.

Postconditions:

- v is now a member of crs-Complete.

CRSClass2

CRSClass1

crs-T 2: crs-VisRel: crs-Visit ⇸ crs-Visit

crs-I 3: Cod(crs-VisRel) ⊆ crs-Proceed

crs-I 4: crs-VisRel⁺ ∩ id[crs-Visit] = ∅

- Inherit the type declarations and invariants of the class CRSClass1

Types:

- crs-VisRel is a tree over crs-Visit

Invariants:

- All members of crs-Visit that are in the codomain of crs-VisRel are in crs-Proceed.
- crs-VisRel is a DAG.

CRSClass2.CreVis(v)

CRSClass2.EmbInOld(V_n,v_o)

crs-Pr 3: V_n: Set[crs-V\crs-Visit]

crs-Pr 4: v_o: crs-Proceed

crs-Pr 5: $\forall v_n : V_n$ • **CRSClass1.CreVis(v_n)**

---

crs-Po 3: RestVisRel' = $(\{v_o\} \cup V_n) \lhd$ crs-VisRel' $\rhd (\{v_o\} \cup V_n)$

crs-Po 4: #RestVisRel' = #V_n

crs-Po 5: Dom(RestVisRel') = V_n

Preconditions:

- V_n is a subset of crs-V that is disjoint wrt crs-Visit (the set of new visit records).
- v_o is a member of crs-Proceed (the existing visit record).
- For each member of the set of new visit records, inherit the preconditions and postconditions of the .CreVis operation

Postconditions:

- RestVisRel is the subset of crs-VisRel that has any of the new and old visit records in its domain or codomain.
- The number of elements in RestVisRel is the same as that in the set of new visit records.
- The domain of RestVisRel is the set of new visit records

NB The postcondition ensures that additions to the tree crs-VisRel comprise only one branch. IE, every member of V_n has one parent, and that is either in V_n or is v_o and v_o has no parent in V_n.

CRSClass2.EmbInNew(V)

crs-Pr 6: V: Set[crs-V\crs-Visit]
crs-Pr 7: #V > 1
crs-Pr 8: ∀v: V • **CRSClass1.CreVis**(v)
crs-Po 6: RestVisRel' = V ◁ crs-VisRel' ▷ V
crs-Po 7: #RestVisRel' = #V-1
crs-Po 8: Cod(RestVisRel') ∪ Dom(RestVisRel') = V

Preconditions:

- V is a subset of crs-V that is disjoint wrt crs-Visit (the set of new visit records).
- V is not the empty set.
- For each member of the set of new visit records, inherit the preconditions anc postconditions of the .CreVis operation

Postconditions:

- RestVisRel is the subset of crs-VisRel that has any of the new visit records in its domain or codomain.
- The number of elements in RestVisRel is one less than the number in V.
- There are no elements in either the domain or codomain of RestVisRel that are not in V, and *vice-versa.*

NB This operation is similar to the previous one except that it does not embed the finished structure in an existing activity, but rather creates a new, isolated single branch of crs-VisRel.

CRSClass2.FinVis(v)

crs-Pr 9: v ∉ Cod(crs-VisRel)
crs-Pr 10: **CRSClass1.FinVis**(v)

Preconditions:

- v is not in the codomain of crs-VisRel
- Inherit the preconditions and postconditions of the .FinVis operation

◇◇◇

Types:

- crs-Pid is a set of members of the carrier set crs-ID.
- crs-VisPid is a total function from the set crs-Visit to the set crs-Pid. Every visit record must be associated with a patient ID record.

Invariants:

- The projection of the crs-VisRel tree into crs-Pid space is a subset of the identity function for crs-Pid. Every visit record is related to the same patient ID record as its parent (through crs-Pid)

CRSClass3.Register(pid)

crs-Pr 11: pid: crs-ID\crs-Pid
crs-Po 9: pid ∈ crs-Pid'

Preconditions:

- pid is a member of crs-ID, but not yet of crs-Pid.

Postconditions:

- pid is now a member of crs-Pid.

CRSClass3.CreVis(v,pid)

crs-Pr 12: pid: crs-Pid
crs-Pr 13: **CRSClass2.CreVis**(v)
crs-Po 10: (v,pid) ∈ crs-VisPid'

Preconditions:

- pid is a registered patient ID - it belongs to the set crs-Pid.
- Inherit the preconditions and postconditions of the CreVis operation.

Postconditions:

- pid is now the patient id associated with the visit record v.

CRSClass3.EmbInOld(V_n,v_o,pid)

crs-Pr 14: pid: crs-Pid
crs-Pr 15: crs-VisPid(v_o) = pid
crs-Pr 16: **CRSClass2.EmbInOld**(V_n,v_o)
crs-Po 11: V_n X {pid} ⊆ crs-VisPid'

Preconditions:

- pid is a registered patient ID - it belongs to the set crs-Pid.
- v_o is associated with the patient ID pid.
- Inherit the preconditions and postconditions of the EmbInOld operation

Postconditions:

- All of V_n are now associated with the patient ID pid via crs-VisPid

CRSClass3.EmbInNew(V,pid)

crs-Pr 17: pid: crs-Pid
crs-Pr 18: **CRSClass2.EmbInNew**(V)
crs-Po 12: V X {pid} ⊆ crs-VisPid'

Preconditions:

- pid is a registered patient ID - it belongs to the set crs-Pid.
- Inherit the preconditions and postconditions of the EmbInNew operation

Postconditions:

- All of V are now associated with the patient ID pid via crs-VisPid

CRSClass3.FinVis(v)

crs-Pr 19: pid: crs-Pid
crs-Pr 20: **CRSClass2.FinVis**(v)

Preconditions:

- pid is a registered patient ID - it belongs to the set crs-Pid.
- Inherit the preconditions and postconditions of the .FinVis operation

◇◇◇

CRSClass4

CRSClass3, CRSTypeClass1

crs-T 7: crs-VisitType: crs-Visit $\rightarrow$ crs-Types

crs-I 7: crs-VisitType ^{o} crs-VisRel ^{o} crs-VisitType^{-1} $\subseteq$ crs-TypeParent

crs-I 8: (im crs-VisitType^{-1}) Cod(crs-TypeParent) $\subseteq$ Cod(crs-VisRel)

- Inherit the type declarations and invariants of the classes CRSClass3 and CRSTypeClass1.

Types:

- crs-VisitType is a total function from crs-Visit to crs-Types

Invariants:

- A projection of the tree crs-VisRel into crs-Type space via the function crs-VisitType is a subset of crs-TypeParent.
- Any visit record that is of a type record that can have a child (through crs-TypeParent) must have such a child visit record.

CRSClass4.CreVis(t,pid→v_n)

crs-Pr 21: t: crs-Types

crs-Pr 22: t ∉ Cod(crs-TypeParent)

. 1

crs-Pr 23: **CRSClass3.CreVis**(pid→ v_n)

crs-Pr 24: t ∉ Dom(crs-TypeParent)

---

crs-Po 13: crs-VisitType'(v_n) = t

. 2

crs-Pr 25: **CRSClass3.EmbInOld**(pid,v_o→ V_n)

crs-Pr 26: v_o ∈ (im crs-VisPid^{-1}) {pid} ∩ (im crs-VisitType^{-1}) (im crs-TypeParent^{+} {t})

crs-Pr 27: (im crs-VisitType) (im crs-VisRel^{-1+}) {v_o} ∩ crs-Proceed ∩ (im crs-TypeParent^{+}) {t} = ∅

crs-Pr 28: #Vn = #((im crs-TypeParent^{+}) {t} ∩ (im crs-TypeParent^{-1*}) {crs-VisitType(v_o)})

crs-Pr 29: v_n ∈ V_n

---

crs-Po 14: (im crs-TypeParent*) {t} \ (im crs-TypeParent* {crs-VisitType(v_o)} = (im crs-VisitType) V_n

crs-Po 15: v_n ∉ Cod(V_n ◁ crs-VisRel' ▷ V_n)

. 3

crs-Pr 30: **CRSClass3.EmbInNew**(pid→V)

crs-Pr 31: (im VisPid^{-1}) {pid} ∩ (im crs-VisitType^{-1}) (im TypeParent^{+} {t}) ∩ crs-Proceed = ∅

crs-Pr 32: #V = #(im TypeParent*) {t}

crs-Pr 33: v_n ∈ V

---

crs-Po 16:(im crs-TypeParent*) {t} = (im crs-VisitType) V

crs-Po 17: v_n ∉ Cod(V ◁ crs-VisRel' ▷ V)

Preconditions:

- t is a record in the set crs-Types
- t is not in the codomain of the tree crs-TypeParent. That is, it does not have any 'children' through crs-TypeParent.

Case 1

Preconditions

- Inherit the preconditions and postconditions of the CreVis operation
- t is not in the domain of crs-TypeParent - thus it does not feature in the tree at all.

Postconditions

- the record type of the visit record vo is now t

Case 2

Preconditions

- Inherit the preconditions and postconditions of the EmbInOld operation.

- Vo is associated with the patient ID pid and is associated with a type record that is an ancestor of type record t (through crs-TypeParent).
- There are no descendants of Vo that fulfil the criteria above and are in the set crs-Proceed.
- The number of elements in Vn is the same as the number of 'ancestors' of t that are also 'descendants' of the type of vo (including vo itself)

Postconditions

- The set of all ancestor types of t, excluding all ancestor types of vo (including the type of vo), is the same as the set of types of Vn.
- vn is not in the codomain of that part of crs-VisRel that is constructed of elements in Vn.

Case 3

Preconditions

- Inherit the preconditions and postconditions of the EmbInNew operation.
- There are no visit records that are of a type that is ancestral to t, that are associated with pid, and are in the set crs-Proceed.
- The number of elements in Vn is the same as the number of 'ancestors' of t.
- vn is in V

Postconditions

- The set of all ancestor types of t is the same as the set of types of V.
- v_n is not in the codomain of that part of crs-VisRel that is constructed of elements in V.

CRSVTClass1.FinVis(v)

◇◇◇

- All members of crs-Types that are not in the domain of crs-TypeParent are to be interpreted as types in Access

◇◇◇

CRSInteraction1.InCreate(A_b,p_n,t_n→a_n)

crs-int-Pr 1: **ATClass2.InCreate**(A_b,p_n,t_n→a_n)

NB This operation is not represented in the IS

CRSInteraction1.Embed(A_p,A_b,t_c→a_c)

crs-int-Pr 2: **ATClass2.Embed**(A_p,A_b,t_c→a_c)

NB This operation is not represented in the IS

CRSInteraction1.SuddenStart(A_p,t_c,a_c)

crs-int-Pr 3: **ATClass2.SuddenStart**(A_p,t_c,a_c)

NB This operation is not represented in the IS

CRSInteraction1.Start(a)

crs-int-Pr 4: **ATClass2.Start**(a)

. 1 crs-int-Pr 5: ActType(a) ∈ Dom(RepT)

crs-int-Pr 6: IntT ⚬ ((Cod({RepT(ActType(a))} ◁ crs-TypeParent*)) ◁ crs-TypeParent) ⚬ RepT ⊆ ActType ⚬ ((Cod({a} ◁ During*)) ◁ During) ⚬ ActType^{-1}

crs-int-Pr 7: **CRSVTClass1.CreVis**(RepT(ActType(a)),RepP(ActSubject(a))→v_n)

crs-int-Po 1: (v_n,a) ∈ IntV'

Preconditions:

- Inherit the preconditions and postconditions of the ATClass2.Start operation

  Case 1

  Preconditions

  - The type of a is represented in the IS
  - The structure that is created as a result of calling .CreVis must be capable of being matched up with an existing structure in the domain that has a as its most junior member.
  - Inherit the preconditions and postconditions of the CRSVTClass1.CreVis operation.

  Postconditions

  - a is now the interpretation of v_n.

CRSInteraction1.Suspend(a)

crs-int-Pr 8: **ATClass2.Suspend**(a)

NB This operation is not represented in the IS

CRSInteraction1.End(a)

crs-int-Pr 9: **ATClass2.End**(a)

. 1 crs-int-Pr 10: **CRS.FinVis**(RepA(a))

Preconditions:

- Inherit the preconditions and postconditions of the ATClass2.End operation

Case 1

Preconditions

- Inherit the preconditions and postconditions of the CRS.FinVis operation

CRSInteraction1.Cancel(a)

crs-int-Pr 11: **ATClass2.Cancel**(a)

NB This operation is not represented in the IS

CRSInteraction1.OutCreate(p,t→a)

crs-int-Pr 12: **ATClass2.OutCreate**(p,t→a)

NB This operation is not represented in the IS

CRSInteraction1.OutEmbed(A_p,t_c→a_c)

crs-int-Pr 13: **ATClass2.OutEmbed**(A_p,t_c→a_c)

NB This operation is not represented in the IS

CRSInteraction1.PatReg(p)

crs-int-Pr 14: **CRSVTClass1.Register**(pid)

crs-int-Pr 15: **ATClass2.PatReg**(p)

crs-int-Po 2: (p,pid) ∈ RepP'

Preconditions:

- Inherit the preconditions and postconditions of the CRSVTClass1.Register operation
- Inherit the preconditions and postconditions of the ATClass2.PatReg operation

Postconditions:

- p is now represented by pid.

CRSInteraction1.PatDereg(p)

crs-int-Pr 16: **ATClass2.PatDereg**(p)

NB This operation is not represented in the IS

CRSInteraction1.PatArrive(p)

crs-int-Pr 17: **ATClass2.PatArrive**(p)

NB This operation is not represented in the IS

CRSInteraction1.PatDepart(p)

crs-int-Pr 18: **ATClass2.PatDepart**(p)

NB This operation is not represented in the IS

CRSInteraction1.PatJoin(a,p)

crs-int-Pr 19: **ATClass2.PatJoin**(a,p)

NB This operation is not represented in the IS

CRSInteraction1.PatLeave(p)

crs-int-Pr 20: **ATClass2.PatLeave**(p)

NB This operation is not represented in the IS

Appendix 5:

A Theory of the DIS1 System and its Interaction with the Domain

This appendix introduces and describes, in an abstract manner, the structure of the proposed first fragment of the Directorate Information System - DIS1. It shows how this is composed of the Outpatient Appointment System (OPAS) and the Clinical Record System (CRS - described in the previous appendix). It then describes how the DIS1 system is interpreted into the domain with the help of an interaction theory.

The classes OPASClock, OPASSlots, OPASConfig, OPASClinics, and OPASAppt describe the OPAS. The classes DIS1TypeClass1 and DIS1Class1 describe how the OPAS and CRS have been mutually constrained to produce the DIS1 system. The classes ClockInteraction, DIS1TypeInteraction1 and DIS1Interaction comprise the interaction theory for the system.

OPASClock

as-T 1: as-FirstT, as-LastT, as-NowT: as-TofD

as-T 2: as-NowD: as-Days

as-T 3: as-Precedes_T, as-Follows_T: as-TofD ⇸ as-TofD

as-T 4: as-Precedes_D, as-Follows_D: as-Days → as-Days

as-T 5: as-Duration: as-TofD x as-TofD ⇸ N

as-I 1: as-Precedes_T^{-1} = as-Follows_T; as-Precedes_D^{-1} = as-Follows_D

as-I 2: as-Precedes_T^{+} ∩ id[as-TofD] = ∅; as-Precedes_D^{+} ∩ id[as-Days] = ∅

as-I 3: Dom(as-Precedes_T) = as-TofD\{as-LastT}

as-I 4: Dom(as-Follows_T) = as-TofD\{as-FirstT}

as-I 5: Dom(as-Duration) = as-Precedes_T^{+}

as-I 6: ∀(t1,t2): Dom(as-Duration) • as-Duration((t1,t2)) = 1+#((im as-Precedes_T) {t1} ∩ (im as-Follows_T) {t2})

as-NowT' = as-FirstT' = 00:00; as-LastT' = 23:59

as-NowD' = 1st January 1994

Types:

NB All the times and the ordering of the times are assumed to be set up by a 'specialisation' operation not described here. This is similar to the specialisation operations of the other theories and puts values into the relevant sets that comply with the invariants.

- as-FirstT, as-LastT, and as-NowT are members of the set as-TofD. This last set represents times of day - as-FirstT and as-LastT the beginning and end of the day respectively, and as-NowT is the current time.
- as-NowD is a member of the set as-Days. This last (infinite) set represents dates - as-NowD the current date.
- as-Precedes_T and as-Follows_T are trees over the set as-TofD.
- as-Precedes_D and as_Follows_D are total trees over the set as-Days. This is possible as as-Days is an infinite set.
- as-Duration is a partial function from the cartesian product of as-TofD with itself to the set of natural numbers. In other words, when supplied with with a pair of records from as-TofD as an

argument, the function will return a number. This number is intended to represent the number of 'ticks' between the times represented by the first and second parts of the pair.

Invariants:

- as-Follows_T is the inverse of as-Precedes_T. as-Follows_D is the inverse of as-Precedes_D.
- as-Precedes_T and as_Precedes_D are directed acyclic graphs. This with the other type declarations and invariants tell us that as-Precedes_T and as-Precedes_D are totally ordered.
- The domain of as-Precedes_T is the set of time records excluding the 'last' time record.
- The codomain of as-Precedes_T is the set of time records excluding the 'first' time record.
- The domain of as-Duration is identical to the transitive closure of as-Precedes_T. This means that pairs in the domain of as-Duration must be such that the first part is 'before' the second part as specified by as-Precedes_T.
- For all pairs in the domain of as-Duration, the returned number is one greater than the intersection of all time records 'after' the first time, and 'before' the second time.

OPASClock.Tick()

. 1	as-Pr 1: as-NowT ≠ as-LastT
	as-Po 1: as-NowT' = as-Precedes_T(as-NowT)
. 2	as-Pr 2: as-NowT = as-LastT
	as-Po 2: as-NowT' = as-FirstT
	as-Po 3: as-NowD' = as-Precedes_D(as-NowD)

Case 1:

Preconditions:

- The current time record is not the time record representing the end of the day

Postconditions:

- The new current time record is the one immediately after the old one as defined by the total arder as-Precedes_T.

Case 2:

Preconditions:

- The current time record is the time record representing the end of the day.

Postconditions:

- The new current time record is the one representing the beginning of the day.
- The new current day record is the one immediately after the old one as defined by the total arder as-Precedes_D.

OPASSlots

OPASClock

as-T 6: as-Slots: Set[as-S]

as-T 7: as-Streams: Set[as-Str]

as-T 8: as-SlotDay: as-Slots → as-Days

as-T 9: as-StreamDay: as-Stream → as-Days

as-T 10: as-SlotStart, as-SlotEnd: as-Slots → as-TofD

as-T 11: as-SlotLength: as-Slots → N

as-T 12: as-SlotStream: as-Slots → as-Streams

as-T 13: as-Precedes_Sl: as-Slots ↛ as-Slots

as-I 7: as-StreamDay ° as-SlotStream ° as-SlotDay^{-1} ⊆ id[as-Days]

as-I 8: as-Precedes_Sl^{+} ∩ id[as-Slots] = ∅

as-I 9: as-SlotStream ° as-Precedes_Sl ° as-SlotStream^{-1} ⊆ id[as-Streams]

as-I 10: as-SlotEnd ° as-SlotStart^{-1} ⊆ as-Precedes_T^{+}

as-I 11: as-SlotLength = as-Duration ° (as-SlotStart ◊ as-SlotEnd)

as-I 12: as-SlotEnd ° as-Precedes_Sl ° as-SlotStart^{-1} ⊆ as-Precedes_T*

as-I 13: #as-Slots\Cod(as-Precedes_Sl) ◁ as-SlotStream = #as-Slots\Dom(as-Precedes_Sl) ◁ as-SlotStream = #Cod(as-SlotStream)

---

as-Slots' = as-Streams' = ∅

- Inherit the type declarations and invariants of the class OPASClock

Types:

- Slots is a subset of as-S. It represents a set of clinic slots
- Streams is a subset of as-Str. It represents the set of 'sessions' which are divided into slots
- Each slot record is associated with a particular date record through the function as-SlotDay. This represents the date when the slot is scheduled to happen.
- Each stream record is associated with a particular date record through the function as-StreamDay. This represents the date when the stream is scheduled to happen.
- Each slot record is associated with a particular time record through the functions as-SlotStart and as-SlotEnd. These represent the start and stop times of the slot.
- Each slot record is associated with a natural number through the function as-SlotLength. This number is intended to represent the duration of the slot.
- Each slot record is associated with a single stream record through the function as-SlotStream.
- Slot records are (partially) ordered through the function as-Precedes_Sl. Thus one slot record can directly precede another.

Invariants:

- A slot record's 'day' is the same as the 'day' of the stream record that the slot record is associated with.

- as-Precedes_Sl is a directed acyclic graph.
- If one slot record precedes another, then they are both associated with the same stream record.
- The 'time' associated with a slot record's start is the same as that associated with its end.
- The length of a slot record is the duration between its 'start' and its 'end'.
- The 'end' of one slot record is always 'before' (according to as-Precedes_Sl) the beginning of the next slot record (or is at the same time).
- For any given stream record, there is only ever one slot record that precedes no other, and one that is preceded by no other.

NB We have not said that all the slot records in a stream are contiguous for a given clinic: gaps such as evenings, days off, illnesses and the like might be represented in the appointment system.

OPASSlots.StreamCreate(d, n→St)

as-Pr 3: d: as-Days; n: N; St: Set[as-Str\as-Streams]
as-Pr 4: #St = n
as-Po 4: St ⊆ as-Streams'
as-Po 5: (im as-StreamDay') St = {d}

Preconditions:

- d is a member of as-Days. n is a natural number. St is a set of stream records that have not yet been created.
- The cardinality of St is n.

Postconditions:

- St is now a set of created streams.
- The day (member of as-Days) that all the stream records are associated with is d.

OPASSlots.StreamCancel(st)

as-Pr 5: st: as-Streams
as-Po 6: st ∉ as-Streams'
as-Po 7: (im as-SlotStream^{-1}) {st} ∩ as-Slots' = ∅

Preconditions:

- st is a single (previously created) stream record.

Postconditions:

- st is no longer a member of as-Streams.
- The slot records that were associated with the stream record through the original as-SlotStream function are no longer in as-Slots.

OPASSlots.SlotsCreate(TN, st → Sl, SlB, SlN, SlE, SlSl)

as-Pr 6: TN: TofD ⇸ N; st: as-Streams

as-Pr 7: Sl: as-S\as-Slots

as-Pr 8: SlB, SlE: as-S\as-Slots ⇸ as-TofD; Sl: Set[as-S\as-Slots]

as-Pr 9: SlN: as-S\as-Slots ⇸ N; SlSl: as-S\as-Slots ⇸ as-S\as-Slots

as-Pr 10: (im as-SlotStream^{-1}) {st} = ∅

as-Pr 11: (as-NowD, as-StreamDay(st)) ∈ as-Precedes_D^{+}

as-Pr 12: Dom(SlB) = Sl; Cod(SlB) = Dom(TN)

as-Pr 13: SlN = (TN ° SlB)$^{-1}$

as-Pr 14: as-Duration ° (SlB ◊ SlE)$^{-1}$ = SlN

as-Pr 15: SlE ° SlB^{-1} ⊆ as-Precedes_T^{+}

as-Pr 16: SlSl^{-1} ∈ as-S\as-Slots ⇸ as-S\as-Slots

as-Pr 17: SlSl^{+} ∩ id[as-S] = ∅

as-Pr 18: SlB ° SlSl ° SlB^{-1} ⊆ as-Precedes_T^{+}

as-Pr 19: #Sl\Dom(SlSl) = #Sl\Cod(SlSl) = 1

---

as-Po 8: SlB ⊆ as-SlotStart'; SlE ⊆ as-SlotEnd'

as-Po 9: SlSl ⊆ as-Precedes_Sl'; SlN ⊆ as-SlotLength'

as-Po 10: Sl X {st} ⊆ as-SlotStream'

as-Po 11: Sl X {as-StreamDay(st)} ⊆ as-SlotDay'

Preconditions:

NB This operation is not as complex as it appears. What it does is set up a number of clinic slot records for a particular stream. Thus most of the parameters are of the same type and will, after the operation, be subsumed within the state components defined in the state schema. The arguments passed are st which is the stream record which is to have slots allocated to it. TN is a function which defines the new slots - their start times and durations. Sl can be thought of as a prototype of a the as-Slots set; SlB and SlE as as-SlotStart and as-SlotStart and as-SlotEnd respectively; SlN as as-SlotLength; and SlSl as as-Precedes_Sl. The preconditions ensure that these 'returned' parameters are such that they can be joined with the main state components according to their intended meaning.

- TN is a partial function from time records to natural numbers. st is a stream record.
- Sl is a set of uncreated slot records.
- SlB and SlE are functions from uncreated slot records to time records. Sl is a set of uncreated set records.
- SlN is a partial function from uncreated slot records to natural numbers. SlSl is a tree over uncreated slot records.
- st has no slot records associated with it yet.
- The current day record must precede that associated with the stream record.
- The domain of SlB is the set of uncreated slots Sl. The codomain of SlB is the time records which are the domain of TN.

- SlN (which will be the duration of slots in Sl) is the composition of TN and SlB (which gives the natural numbers associated with the slots in the domain of SlB).
- SlN is equal to the slots in Sl associated with the number which represent their duration as deduced from SlB (the beginning of the slots), SlE (the end of the slots) and as-Duration which enable us to turn pairs of time records into natural numbers representing the duration of the gap between them.
- The time record associated with a slot in SlB must be 'before' the time record associated with the same slot in SlE.
- The inverse of SlSl is a tree over uncreated slot records. This means, assuming that SlSl is to represent the 'ordering' of slot records, that a slot record can be after at most one other, and before at most one other.
- SlSl is a directed acyclic graph.
- The 'beginnings' of pairs of slots in SlSl must be ordered - the 'beginning' of the first slot record must be 'before' (as defined by as-Precedes_T) the 'beginning' of the second.
- There is one slot record in Sl that is not in the domain of SlSl - similarly there is one slot record in Sl that is not in the codomain of SlSl. This final precondition together with the others means that SlSl must be represent a totally ordered set (or at least $SlSl^+$ must be - depending on the definition of total ordering used): it is a single chain of uncreated slot records.

Postconditions:

- SlB is now is as-SlotStart. SlE is now is as-SlotEnd.
- SlSl is now in the ordering tree as-Precedes_Sl. SlN is now in the function as-SlotLength.
- All of Sl is now associated with the stream record st through the function as-SlotStream.
- All of Sl is now, through as-SlotDay, associated with the day record that st was associated with through as-StreamDay.

NB This operation schema means that we must create all the slot records for a given stream record 'in one go'.

◇◇◇

- as-CTypes is a set of elements of type CT. This records the types of clinics supported by this specialisation of the application.
- as-ApptMode is a set of elements of type M. This records the 'modes' that a particular clinic supports - this might be new visit, followup, review and so on.
- as-CTypeModes is a relation between clinc type records and appointment mode records. Certain clinic type support certain appointment modes - not all appointment modes are supported by all clinics however (specialist nurses do not have 'review' visits.
- as_CModeLength is a function from pairs in as-CTypeModes to natural numbers. These numbers represent the duration that a visit to a particular clinic, of a particular mode, is scheduled to last.

◇◇◇

- The same consultant record cannot be associated with two clinic sessions on the same day.

OPASClinics.ClinicSetUp(ct, D, Dn →Cl)

as-Pr 20: D: Set[as-Days]; Dn: as-Days ⇸ N; ct: as-CType; Cl: Set[C\as-Clinics]

as-Pr 21: Dom(Dn) = D

as-Pr 22: ∀d: D • **OPASSlots.StreamCreate**(d,Dn(d)→St)

---

as-Po 12: Cl ⊆ as-Clinics'

as-Po 13: Cl X {ct} ⊆ as-ClinicType'

as-Po 14: (im as-ClinicDay') Cl = D

as-Po 15: ∀cl: Cl • #(im as-StreamClinic'$^{-1}$) {cl} = Dn(as-ClinicDay(cl))

as-Po 16: (im as-StreamClinic'$^{-1}$) {Cl} ⊆ as-Streams'\as-Streams

Preconditions:

NB This operation 'sets up' a clinic. The clinic type, days, and numbers ascociated with those days are provided as parameters and a set of clinic session records is returned. A clinic session is created for each of the days, and the number of newly created streams, given by the appropriate number in Dn, is associated with each clinic session.

- D is a set of day records. Dn is a partial function from day records to natural numbers. ct is a clinic type record. Cl is a set of uncreated clinic session records.
- The domain of Dn is D.
- For all day records in D, Inherit the preconditions and postconditions of the StreamCreate operation (taking each member of D as an argument).

Postconditions:

- Cl is now a set of clinic session records.
- All the records in Cl are now associated with the clinic type record ct throught the function as-ClinicType.
- The records in Cl are now associated with day records in D.
- The number of stream records associated with a given clinic session record is given by the number returned by Dn when given the day record associated with that clinic session as an argument.
- The streams now associated with clinic session records in Cl are all new stream records.

OPASClinics.StreamCancel(st)

as-Pr 23: **OPASSlots.StreamCancel**(st)

Preconditions:

- Inherit the preconditions and postconditions of the StreamCancel operation

OPASClinics.SlotsCreate(TAm, st → Sl, SlAm, SlB, SlN, SlE, SlSl)

as-Pr 24: **OPASSlots.SlotsCreate**(TN, st → Sl, SlB, SlN, SlE, SlSl)

as-Pr 25: TAm: as-TofD ⇸ as-ApptMode

as-Pr 26: Cod(TAm) ⊆ (im as-CTypeModes) {as-ClinicType(as-StreamClinic(st))}

as-Pr 27: TN = (as-CModeLength ° (Dom(TAm) x {as-ClinicType(as-StreamClinic(st))}) ◊ TAm))

as-Pr 28: SlAm = (TAm ° SlB)$^{-1}$

as-Po 17: SlAm ⊆ as-SlotMode'

Preconditions:

NB This operation is similar to the earlier SlotCreate one except that the parameter TAm is passed which gives an appointment mode for each time. From this the start of the sessions, and their length can be deduced, hence giving us the TN of the previous schema. The operation allows us to set up all the slots for a stream at a time.

- Inherit the preconditions and postconditions of the SlotsCreate operation.
- TAm is a partial function from time records to appointment mode records.
- The appointment mode records in the codomain of the parameter TAm must be valid appointment modes for the clinic type of the clinic session record of the stream record st.
- TN is a function from time records to numbers calculated by taking a time record from TAm and associating it with a number derived from the function as-CModeLength. To this end the function is passed a pair: the clinic type of the clinic session record associated with the stream st, followed by the appointment mode - from this pair and the function as-CModeLength the standard duration of the slot can be calculated and passed to the earlier operation as TN.
- SlAm is a function which takes the (uncreated) slot record in the domain of SlB and associates it with the appointment mode linked to the same time in TAm as the slot is in SlB.

Postconditions:

- SlAm is now a subset of as-SlotMode

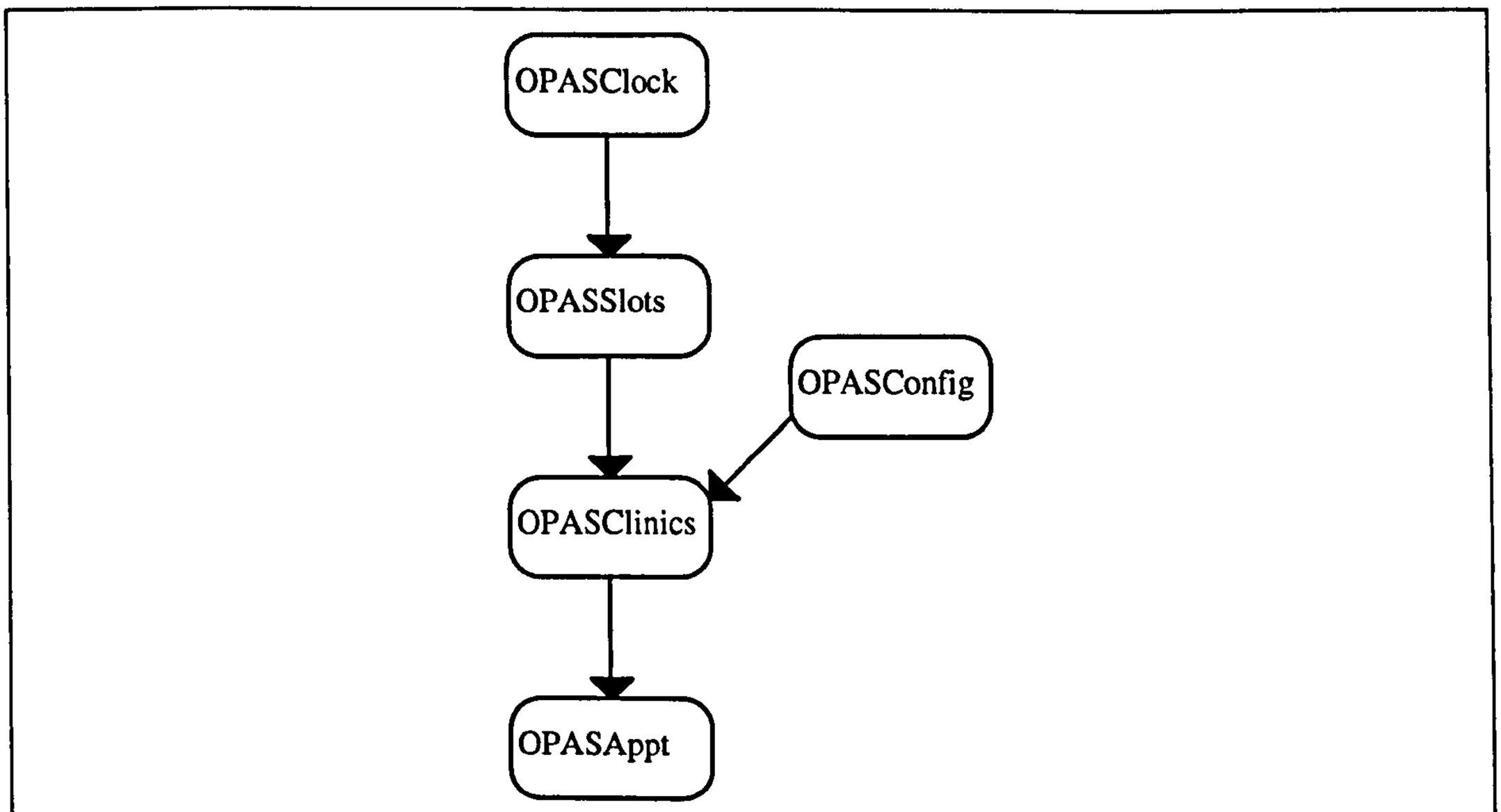


Figure App5-1: Class structure of the first five classes in OPAS

OPASAppt

OPASClinics

as-T 23: as-Pid: Set[as-P]

as-T 24: as-Appointment: as-Slots ⇸ as-Pid

as-T 25: as-ArrivalTime, as-StartTime, as-EndTime, as-ApptTimes: as-Slots ⇸ as-TofD

as-I 18: Dom(as-ArrivalTime) ⊆ Dom(as-Appointment)

as-I 19: Dom(as-StartTime) ⊆ Dom(as-ArrivalTime)

as-I 20: Dom(as-EndTime) ⊆ Dom(as-StartTime)

as-I 21: Cod(as-Appointment) = as-Pid

as-I 22: as-ApptTimes = as-ArrivalTime ∪ as-StartTime ∪ as-EndTime

as-I 23: Cod(Dom(as-ApptTimes) ◁ as-SlotDays) ⊆ (im as-Precedes_D*) { as-NowD}

as-I 24: Cod(Dom(as-SlotDays ▷ { as-NowD}) ◁ as-ApptTimes) ⊆ (im as-Precedes_T*) { as-NowT}

as-Pid' = ∅

- Inherit the type declarations and invariants of the class OPASClinics

Types:

- as-Pid is a set of records of type as-P. A record in this set is a patient identifier and hence represents a patient.
- as-Appointment is a partial function from slot records to patient identifiers. This is intended to represent appointments.

- as-ArrivalTime, as-StartTime, as-EndTime, and as-ApptTimes are all partial functions from slot records to time records. These functions are intended to represent patient arrival time, appointment start time, appointment end time, and the union of all those functions respectively.

Invariants:

- The slot records in the domain of as-ArrivalTime are all in the domain of as-Appointment.
- The slot records in the domain of as-StartTime are all in the domain of as-ArrivalTime.
- The slot records in the domain of as-EndTime are all in the domain of as-StartTime.
- All members of the codomain of as-Appointment are patient identifiers.
- as-AppTimes is defined to be the union of as-ArrivalTime, as-StartTime, and as-EndTime.
- All the day records that are associated with slot records that have an entry in as-ApptTimes must either be 'before' or equal to the 'current' day record.
- All the time records that are associated with slot records that have an entry in as-ApptTimes and are linked to the 'current' day record must be equal to or 'before' the current time record.

OPASAppt.BookKnownP(pid,sl)

as-Pr 29. pid: as-Pid
as-Pr 30: sl: as-Slots\Dom(as-Appointment)
as-Pr 31: as-SlotDay(sl) ∈ (im as-Precedes_D$^+$)[as-NowD]
as-Po 18: (sl, pid) ∈ as-Appointment'

Preconditions:

- pid is a patient identifier.
- sl is a slot record that has not yet been assigned a patient identifier.
- The day record that sl is assciated with is 'after' the 'current' one.

Postconditions:

- sl is now associated with pid through the function as-Appointment

OPASAppt.BookUnknownP(sl→pid)

as-Pr 32: pid: as-P\as-Pid
as-Pr 33: sl: as-Slots\Dom(as-Appointment)
as-Pr 34: as-SlotDay(sl) ∈ (im as-Precedes_D$^+$)[as-NowD]
as-Po 19: pid ∈ as-Pid'
as-Po 20: (sl, pid) ∈ as-Appointment'

Preconditions:

- pid is an uncreated patient identifier
- sl is a slot record that has not yet been assigned a patient identifier.
- The day record that sl is assciated with is 'after' the 'current' one.

Postconditions:

- pid is now a patient identifier.
- sl is now associated with pid through the function as-Appointment

OPASAppt.ChangeBooking(sl_o, sl_n)

as-Pr 35: sl_o, sl_n: Dom(as-SlotDay ▷ (im as-Precedes_D$^+$ {as-NowD}))

as-Pr 36: $sl_o \in$ Dom(as-Appointment)

as-Pr 37: $sl_n \notin$ Dom(as-Appointment)

as-Po 21: $sl_o \notin$ as-Appointment'

as-Po 22: (sl_n, as-Appointment(sl_o)) $\in$ as-Appointment'

Preconditions:

NB This operation changes an appointment record insofar as it removes a patient identifier from association with one slot record and associates it with another.

- sl_o (old slot record) and sl_n (new slot record) are both slot records that are associated with day records that are 'after' the 'current' one.
- sl_o is a slot record that is associated with a patient identifier through as-Appointment.
- sl_n is a slot record that is not associated with a patient identifier through as-Appointment (we can thus see that sl_o and sl_n must be distinct).

Postconditions:

- sl_o is no longer associated with a patient identifier through as-Appointment.
- sl_n is now associated with a patient identifier through as-Appointment. The patient identifier is the same as the one that sl_o was associated with before the operation.

OPASAppt.CancelBooking(sl)

as-Pr 38: sl: Dom(as-SlotDay ▷ (im as-Precedes_D$^+$ {as-NowD})) ∩ Dom(as-Appointment)

as-Po 23: sl $\notin$ as-Appointment'

Preconditions:

- sl is a slot record that is associated with a day record that is 'after' the 'current' one, and is also associated with a patient id via the as-Appointment relation.

Postconditions:

- sl is no longer associated with a patient id via the relation as-Appointment

OPASAppt.StreamCancel(st)

as-Pr 39: st $\in$ Dom(as-StreamDay ▷ (im as-Precedes_D$^+$ {as-NowD}))

as-Pr 40: **OPASClinics.StreamCancel**(st)

as-Po 24: (im as-SlotStream^{-1}){st} ∩ Dom(as-Appointments') = ∅

Preconditions:

- st is a stream record that is associated with a day record that is 'after' the 'current' one.

- Inherit the preconditions and postconditions of the StreamCancel operation

Postconditions:

- None of the slots associated with the stream record st through the as-SlotStream relation is now associated with a patient id via the relation as-Appointment

OPASAppt.PatArrive(sl,τ)

as-Pr 41: sl: (Dom(as-SlotDay ▷ {as-NowD}) \ Dom(as-ArrivalTime)) ∩ Dom(as-Appointment)
as-Po 25: (sl,τ) ∈ as-ArrivalTime'

Preconditions:

- sl is a slot record that is associated with a day record that is 'after' the 'current' one, and is also associated with a patient id via the as-Appointment relation, but is not a slot record that has an 'arrival' time recorded for it.

Postconditions:

- The time record τ is now associated with the slot record sl through the relation as-ArrivalTime.

NB This operation is supplied with τ as an argument so that details of when the patient arrived can be entered after the event.

OPASAppt.ApptStart(sl)

as-Pr 42: sl: (Dom(as-SlotDay ▷ { as-NowD}) \ Dom(as-StartTime)) ∩ Dom(as-ArrivalTime)
as-Po 26: (sl, as-NowT) ∈ as-StartTime'

Preconditions:

- sl is a slot record that: is associated with a day record that is 'after' the 'current' one; has had an 'arrival' time record associated with it; and has not had a 'start' time record associated with it.

Postconditions:

- The current time record is now associated with the slot record sl through the relation as-StartTime

OPASAppt.ApptEnd(sl)

as-Pr 43: sl: (Dom(as-SlotDay ▷ { as-NowD}) \ Dom(as-EndTime)) ∩ Dom(as-StartTime)
as-Po 27: (sl, as-NowT) ∈ as-EndTime'

Preconditions:

- sl is a slot record that: is associated with a day record that is 'after' the 'current' one; has had a 'start' time record associated with it; and has not had an 'end' time record associated with it.

Postconditions:

- The current time record is now associated with the slot record sl through the relation as-EndTime

OPASAppt.ClinicSetup(ct,D,Dn→Cl)

as-Pr 44: **OPASClinics.ClinicSetup**(ct,D,Dn→Cl)

Preconditions:

- Inherit the preconditions and postconditions of the ClinicSetup operation

OPASAppt.SlotsCreate(TAm,st)

as-Pr 45: **OPASClinics.SlotsCreate**(TAm,st→Sl,SlAm,SlB,SlN,SlE,SlSl)

Preconditions:

- Inherit the preconditions and postconditions of the SlotsCreate operation

◇◇◇

DIS1Class1

DIS1TypeClass1, OPASAppt, CRSClass4

dis-T 3: dis-Activities, dis-Request: Set[crs-V]

dis-T 4: dis-VisRel: dis-Activities ⇸ dis-Activities

dis-T 5: dis-Clist: Set[dis-Cl]

dis-T 6: dis-ActSubject: dis-Activities → crs-Pid; dis-ActType: dis-Activities → crs-Types

dis-T 7: dis-ActSlot: dis-Activities ⇸ as-Slots; dis-Pidas: crs-Pid ⇸ as-Pid

dis-T 8: dis-StreamClist: as-Streams → dis-Clist

dis-I 5: dis-Request = dis-Activities \ crs-Visits

dis-I 6: crs-Visits ◁ dis-ActSubject = crs-VisPid; crs-Visits ◁ dis-ActType = crs-VisitType

dis-I 7: Cod(crs-VisRel) ⊆ crs-Proceed

dis-I 8: crs-VisRel ⊆ dis-VisRel ∧ dis-VisRel^{+} ∩ id[dis-Activity] = ∅

dis-I 9: dis-ActType ⚬ dis-VisRel ⚬ crs-VisitType^{-1} ⊆ crs-TypeParent

dis-I 10: (im dis-ActType^{-1}) Cod(crs-TypeParent) ⊆ Cod(dis-VisRel)

dis-I 11: Dom(dis-VisRel\crs-VisRel) ⊆ dis-Request ∧ Cod(dis-VisRel\crs-VisRel) ∩ crs-Proceed ⊆ Cod(crs-VisRel)

dis-I 12: dis-ActSlot^{-1} ∈ Dom(as-Appointments) → dis-Activities

dis-I 13: dis-Pidas^{-1} ∈ as-Pid → crs-Pid

dis-I 14: (dis-StreamClist ◊ as-StreamDay)$^{-1}$ ∈ (dis-Clist X as-Day ⇸ as-Stream)

dis-I 15: ClinicType ⚬ StreamClinic ⚬ StreamClist ∈ dis-Clist ⇸ as-Ctypes

dis-I 16: Dom(as-Appointments) ◁ ((as-ClinicType ⚬ as-StreamClinic ⚬ as-SlotStream) ◊ as-SlotMode) = dis-TypeLink ⚬ dis-ActType ⚬ dis-ActSlot^{-1}

dis-I 17: as-Appointment ⚬ dis-ActSlot ⚬ dis-ActSubject^{-1} ⊆ dis-Pidas

dis-I 18: (im dis-ActType) Dom(dis-ActSlot) ⊆ dis-Bookable

dis-I 19: dis-ActType ⚬ dis-VisRel ⚬ dis-ActType^{-1} ⊆ crs-TypeParent

dis-I 20: #(dis-Request ◁ dis-ActSubject) = #(dis-Request ◁ dis-ActSubject) ⚬ dis-ActType^{-1}

dis-I 21: #(crs-Proceed ◁ crs-VisPid) = #(crs-Proceed ◁ crs-VisPid) ⚬ crs-VisitType^{-1}

dis-I 22: (im dis-ActSlot^{-1}) Dom(as-SlotStart) ⊆ crs-Visits

---

dis-Activities' = ∅

Types:

- The sets dis-Activities and dis-Request are of the same type as crs-Visits. dis-Activities is (loosely) a record of activities; dis-Request (loosely) of requests.
- The relation dis-VisRel is a tree over dis-Activities.
- The set dis-Clist stores the DIS representation of clinic lists
- dis-ActSubject links every activity record with a patient id: dis-ActType links every activity record with a (visit) type from the CRS

- dis-ActSlot is a partial function that returns, for any activity record (that is in the function), the corresponding slot record from OPAS: dis-Pidas is a function that returns, for any patient id in the CRS, the corresponding patient id in OPAS.
- dis-StreamClist is a function that returns, for any stream record it is given, the corresponding clinic list record in DIS.

Invariants

- dis-Request is all activity records that are stored in the DIS but not in the CRS.
- The function crs-Vispid is the same as dis-ActSubject for all non-requests; the function crs-VisitType is the same as dis-ActType for all non-requests.
- The codomain of crs-VisRel is a subset of crs-Proceed. 'Parent' visit records cannot be completed - only those that can have no children may be. Similarly request records cannot be members of crs-Visit and so cannot be in crs-VisRel.
- crs-VisRel is a subset of dis-VisRel and dis-VisRel is a directed acyclic graph.
- A projection of the tree dis-VisRel into crs-Type space via the function dis-ActType is a subset of crs-TypeParent.
- Any activity record that is of a type record that can have a child (through crs-TypeParent) must have such a child activity record. If the parent is a visit record, then so must the child be: if it the parent is a request record then so must the child be.
- The domain of the difference between dis-VisRel and crs-VisRel must be a subset of dis-Request (if it has started it is a visit record and so therefore is its parent meaning that the pair is in crs-VisRel); if an activity record in the codomain has started then there must be a corresponding child for that activity record in crs-VisRel.
- The inverse of dis-ActSlot is a total function from all slot records associated with patient ids to dis-Activities.
- The inverse of dis-Pidas is a total function - for any patient id in OPAS, there is exactly one corresponding one in DIS.
- Each clinic list has at most one stream associated with it on any one day
- A clinic list is associated with one clinic type only.
- The clinic type and mode record for a particular slot record that has a patient id associated with it is the same as the value obtained by taking the clinic type and mode record linked to the type of the activity record associated with that slot.
- The pair (p_{dis},p_{as}) where p_{dis} is the patient id associated (in the DIS) with an activity record, and p_{as} is the patient id associated (in the OPAS) with the slot record which is related to that activity record, are in the relation dis-Pidas.
- The type of any activity that is related to a slot is a bookable type.
- Only activity record hierarchies allowable through TypeParent are allowed (cf case for CRS).
- Only one request record of a particular type is allowed for one patient id.
- Only one proceeding activity record of a particular type is allowed for one patient id.
- Any slot record that has 'started' corresponds to an activity that has started - ie a member of crs-Visits.

DIS1Class1.Create(p_n,t_n→a_n)

dis-Pr 1: t_n: dis-Accessible; a_n: crs-V\dis-Activities; p_n: crs-Pid

dis-Pr 2: (im dis-ActType^{-1}) $\{t_n\} \cap$ (dis-ActSubject^{-1}) $\{p_n\} \cap$ dis-Request = $\varnothing$

dis-Po 1: $a_n \in$ dis-Request'

dis-Po 2: $(a_n,p_n) \in$ dis-ActSubject'

dis-Po 3: $(a_n,t_n) \in$ dis-ActType'

Preconditions:

- t_n is an accessible type: a_n is not yet an activity: p_n is a registered patient id.
- There is currently no request of type t_n that has p_n as its subject

Postconditions:

- a_n is now a request record.
- a_n is now asociated with the patient id p_n through the function dis-ActSubject.
- a_n is now asociated with the activity type record t_n through the function dis-ActType.

DIS1Class1.SuddenStart(t_c,pid→a_c)

dis-Pr 3: t_c: dis-Unplanned; pid: crs-Pid

dis-Pr 4: (im dis-ActType^{-1}) dis-Accessible $\cap$ (im dis-ActSubject^{-1}) {pid} $\cap$ (im dis-ActType^{-1}) (im crs-TypeParent^{+}) $\{t_c\} \neq \varnothing$

dis-Pr 5: (im dis-ActType^{-1}) $\{t_c\} \cap$ (im dis-ActSubject^{-1}) {pid} $\cap$ crs-Proceed = $\varnothing$

dis-Pr 6: **CRSClass4.CreVis**(t_c,pid→a_c)

Preconditions:

- t_c is an unplanned activity; pid is a registered patient id.
- There is an activity record that can be a parent of the new activity record and is in dis-Accessible.
- There are no activity records of the same type as the new one that are associated with the same patient id that are in the set crs-Proceed.
- Inherit the preconditions and postconditions of the CreVis operation

DIS1Class1.Generate(t_c,pid→a_c)

dis-Pr 7: t_c: crs-Types; pid: crs-Pid

dis-Pr 8: a_c: crs-V\dis-Activities; A_n: Set[crs-V\dis-Activities]

dis-Pr 9: a_p: dis-Activities\crs-Complete ∩ (im dis-ActSubject^{-1}) {pid} ∩ (im dis-ActType^{-1}) (im crs-TypeParent^{+}) {t_c}

dis-Pr 10: (im dis-ActType^{-1}) {t_c} ∩ (im dis-ActSubject^{-1}) {pid} ∩ dis-Request = ∅

dis-Pr 11: ~∃a: dis-Activities\ crs-Complete ∩ (im dis-ActType^{-1}) (im crs-TypeParent^{+}) {t_c} • (a,a_p) ∈ dis-VisRel^{+}

dis-Pr 12: #A_n = #((im crs-TypeParent^{+}) {t_c}\ (im crs-TypeParent*) {dis-ActType(a_p)})

---

dis-Po 4: {a_c} ∪ A_n ⊆ dis-Request'

dis-Po 5: (im dis-VisRel') {a_c} ∪ A_n = A_n ∪ {a_p}

dis-Po 6: dis-ActType' ° ({a_c,a_p} ∪ A_n ◁ dis-VisRel' ▷ {a_c,a_p} ∪ A_n) ° dis-ActType'$^{-1}$ ⊆ crs- TypeParent

dis-Po 7: (a_c,t_c) ∈ dis-ActType'

dis-Po 8: (im dis-ActSubject') {a_c} ∪ A_n = {pid}

Preconditions:

- t_c is a type record; pid is a patient id.
- a_c is an uncreated activity record; A_n is a set of uncreated activity records.
- a_p is an uncomplete activity record that is associated with the same patient id and is a possible parent of the new activity record (through crs-TypeParent).
- There are no activity records of the same type as the new one that are associated with the same patient id that are in the set dis-Request.
- There is no uncompleted activity record that is a possible parent of the new activity record and is a child of a_p
- The number of elements in A_n is the same as the number of 'ancestors' of tc that are also 'descendants' of the type of a_p (including a_p itself).

Postconditions:

- All the newly created activity records (all of a_c and A_n) are now requests.
- The parent activity record of any of the newly created record is either in A_n or the parent activity record a_p.
- The projection of that part of dis-VisRel which contains domain and codomain purely from any of the activities referred to in the operation onto the set of type records is a subset of crs-TypeParent.
- a_c is now of the type t_c.
- The patient id associated with all the newly created activity records is pid.

DIS1Class1.Book(t_c,pid,c,d,$\tau_b \rightarrow a_c$,s)

dis-Pr 13: pid: crs-Pid; t_c: dis-Bookable; c: dis-Clists; d: (im as-Precedes_D$^+$ {as-NowD}); τ_b: TofD

dis-Pr 14: a_c: crs-V\dis-Activities; A_n: Set[crs-V\dis-Activities]

dis-Pr 15: a_p: dis-Activities\Complete $\cap$ (im dis-ActSubject^{-1}) {pid} $\cap$ (im dis-ActType^{-1}) (im crs-TypeParent$^+$) {t_c}

dis-Pr 16: (im dis-ActType^{-1}) {t_c} $\cap$ (im dis-ActSubject^{-1}) {pid} $\cap$ dis-Request = $\varnothing$

dis-Pr 17: $\sim\exists$a: dis-Activities\crs-Complete $\cap$ (im dis-ActType^{-1}) (im crs-TypeParent$^+$) {t_c} $\bullet$ (a,a_p) $\in$ dis-VisRel$^+$

dis-Pr 18: #A_n = #((im crs-TypeParent$^+$) {t_c}\ (im crs-TypeParent*) {dis-ActType(a_p)})

dis-Pr 19: s $\in$ (im as-SlotStream^{-1}) {(dis-StreamClist $\Diamond$ as-StreamDay)$^{-1}$ (c,d)}

dis-Pr 20: s $\in$ (as-SlotStart^{-1}) {τ_b}

dis-Pr 21: dis-TypeLink(t_c) = ((as-ClinicType $\circ$ as-StreamClinic $\circ$ as-SlotStream) $\Diamond$ as-SlotMode) (s)

dis-Pr 22: pid $\in$ Dom(dis-Pidas) $\Rightarrow$ aspid = dis-Pidas(pid) $\wedge$ **OPASAppt.BookKnownP**(dis-Pidas(pid),s)

dis-Pr 23: pid $\notin$ Dom(dis-Pidas) $\Rightarrow$ **OPASAppt.BookUnknownP**(s$\rightarrow$aspid)

---

dis-Po 9: {a_c} $\cup$ A_n $\subseteq$ dis-Request'

dis-Po 10: (im dis-VisRel') {a_c} $\cup$ A_n = A_n $\cup$ {a_p}

dis-Po 11: dis-ActType' $\circ$ ({a_c,a_p} $\cup$ A_n $\lhd$ dis-VisRel' $\rhd$ {a_c,a_p} $\cup$ A_n) $\circ$ dis-ActType'$^{-1}$ $\subseteq$ crs-TypeParent

dis-Po 12: (a_c,t_c) $\in$ dis-ActType'

dis-Po 13: (im dis-ActSubject') {a_c} $\cup$ A_n = {pid}

dis-Po 14: (a_c,s) $\in$ dis-ActSlot'

dis-Po 15: (pid,aspid) $\in$ dis-Pidas'

Preconditions:

- pid is a patent id; t_c is a type record from dis-Bookable; c is a clinic list record; d is a day record after the current one; τ_b is a time of day.
- a_c is an uncreated activity record; A_n is a set of uncreated activity records.
- a_p is an uncomplete activity record that is associated with the same patient id and is a possible parent of the new activity record (through crs-TypeParent).
- There are no activity records of the same type as the new one that are associated with the same patient id that are in the set dis-Request.
- There is no uncompleted activity record that is a possible parent of the new activity record and is a child of a_p
- The number of elements in A_n is the same as the number of 'ancestors' of tc that are also 'descendants' of the type of a_p (including a_p itself).
- s is a slot that is associated with a stream that is in its turn associated with the clinic list recoird c and the day record d.
- The 'start' of s is τ_b.

- The clinic type and mode pair returned by dis-TypeLink when it is passed tc as an argument is the same as that obtained from knowing the clinic type of the clinic list associated with the stream associated with the slot, and the mode associated with the slot.
- If pid is known to OPAS and associated with an appointment then aspid is the appropriate OPAS patient id (ie that linked with pid through dis-Pidas) and inherit the preconditions and postconditions of the **BookKnownP** operation.
- If pid is not known to OPAS and associated with an appointment then inherit the preconditions and postconditions of the **BookUnknownP** operation.

Postconditions:

- All the newly created activity records (all of a_c and A_n) are now requests.
- The parent activity record of any of the newly created record is either in A_n or the parent activity record a_p.
- The projection of that part of dis-VisRel which contains domain and codomain purely from any of the activities referred to in the operation onto the set of type records is a subset of crs-TypeParent.
- a_c is now of the type t_c.
- The patient id associated with all the newly created activity records is pid.
- a_c is now associated with the slot s through the relation dis-ActSlot.
- pid and aspid are now related to each other via dis-Pidas.

DIS1Class1.Schedule(a,c,d,$\tau_b \rightarrow$s)

dis-Pr 24: a: dis-Request

dis-Pr 25: c: dis-Clists; d: (im as-Precedes_D$^+$) {as-NowD}; τ_b: as-TofD

dis-Pr 26: dis-ActType(a) $\in$ dis-Bookable

dis-Pr 27: s $\in$ (im SlotStream^{-1}) {(dis-StreamClist $\Diamond$ as-StreamDay)$^{-1}$ (c,d)}

dis-Pr 28: s $\in$ (SlotStart^{-1}) {t_b}

dis-Pr 29: dis-TypeLink(dis-ActType(a)) =
((as-ClinicType $\circ$ as-StreamClinic $\circ$ as-SlotStream)$\Diamond$ as-SlotMode) (s)

dis-Pr 30: **OPASAppt.MakeBooking**(dis-Pidas(dis-ActSubject(a)),s)

---

dis-Po 16: (a,s) $\in$ dis-ActSlot'

Types

- a is a request record
- c is a clinic list; d is a day after today; τ_b is a time of day

Preconditions

- The type of a is in dis-Bookable
- s is a associated with a stream that is associated with clinic list c and takes place on day d
- s is a slot that starts at time τ_b
- The clinic type and appointment mode of s have the same values as those associated with t_c through TypeLink
- then inherit the preconditions and postconditions of the **MakeBooking** operation.

Postconditions

- s is now the slot associated with a_c

DIS1Class1.Unbook(a)

dis-Pr 31: a: Dom(dis-ActSlot)
dis-Pr 32: **OPASAppt.CancelBooking**(dis-ActSlot(a))
dis-Po 17: (a,s) ∉ dis-ActSlot'

Types

- a is an activity record associated with a slot record through dis-ActSlot

Preconditions

- then inherit the preconditions and postconditions of the **CancelBooking** operation

Postconditions

- s is no longer associated with a_c through dis-ActSlot

DIS1Class1.Start(a)

dis-Pr 33: a: dis-Request
dis-Pr 34: **CRSClass4.CreVis**(dis-ActType(a),dis-ActSubject(a)→a)
dis-Pr 35: a ∈ Dom(dis-ActSlot) ⇒ **OPASAppt.ApptStart**(dis-ActSlot(a))

Preconditions

- a is a request record
- Invoke **CRSClass4.CreVis**
- If a is associated with a slot record through dis-ActSlot, then inherit the preconditions and postconditions of the **ApptStart** operation.

DIS1Class1.End(a)

dis-Pr 36: **CRSClass4.FinVis**(a)
dis-Pr 37: a ∈ Dom(dis-ActSlot) ⇒ **OPASAppt.ApptEnd**(dis-ActSlot(a))

Preconditions

- Invoke **CRSClass4.FinVis**
- If a is associated with a slot record through dis-ActSlot, then then inherit the preconditions and postconditions of the **ApptEnd** operation.

DIS1Class1.Cancel(a)

dis-Pr 38: a: dis-Request
dis-Po 18: a ∉ dis-Activities

Preconditions

- a is a request record

Postconditions

- a is no longer an activity record

DIS1Class1.PatArrive(pid,τ)

dis-Pr 39: pid: crs-Pid

dis-Pr 40: ∀s: (as-Appointment^{-1}) {dis-Pidas(pid)} ∩ (im SlotDay^{-1}){as-NowD} • **OPASAppt.PatArrive**(s,τ)

Types

- pid is the id of a registered patient

Preconditions

- For every slot record associated with patient id pid on todays date, inherit the preconditions and postconditions of (a separate instance of) the **PatArrive** operation.

DIS1Class1.Register(pd→pid)

dis-Pr 41: **CRSClass4.Register**(pd→pid)

Preconditions

- Inherit the preconditions and postconditions of the **CRSClass4** operation.

DIS1Class1.ClinicSetup(ct,D,Dn,DClist→Cl)

dis-Pr 42: DClist: as-Days ⇿ dis-Cl

dis-Pr 43: Dom(DClist) = Dom(Dn) = D

dis-Pr 44: ∀d: D • # (im DClist) {d} = Dn(d)

dis-Pr 45: **OPASAppt.ClinicSetup**(ct,D,Dn→Cl)

dis-Po 19: Cod(DClist) ⊆ dis-Clist'

dis-Po 20: Cod(dis-StreamClist ⩥ (as-Streams' \ as-Streams)) = Cod(DClist)

dis-Po 21: ∀d: D, ∀list: Cod(DClist) • #(im as-StreamClist'$^{-1}$) {list} ∩ (im as-StreamDay'$^{-1}$) {d} ≤ 1

dis-Po 22: (im as-ClinicType' ⚬ as-StreamClinic' ⚬ as-StreamClist') Cod(DClist) = {ct}

Preconditions:

- DClist is a relation between day records and clinic list records (which may be created or uncreated).
- Each day record in D is represented in the domain of DClist and the domain of Dn
- For any day record in D, there are as many clinic list records associated with that day as there are stream records (through Dn).
- Inherit the preconditions and postconditions of the **ClinicSetup** operation.

Postconditions:

- All clinic list records referred to in the argument are now in dis-Clist

- Stream records created as a result of this operation are associated with the clinic list records referred to in the operation's parameters.
- On any given day, a given clinic list record can have at most one stream record associated with it.
- All clinic list records referred to in the operation's parameters are associated with stream records that are in turn associated with clinic records that are all of type ct.

DIS1Class1.StreamCancel(st)

dis-Pr 46: **OPASAppt.StreamCancel**(st)

Preconditions:

- Inherit the preconditions and postconditions of the **StreamCancel** operation from the OPASAppt class.

DIS1Class1.SlotsCreate(TAm,st)

dis-Pr 47: **OPASAppt.SlotsCreate**(TAm,st)

Preconditions:

- Inherit the preconditions and postconditions of the **SlotsCreate** operation from the OPASAppt class..

◇◇◇

- RepTime is a total function from time in the domain to time records in the OPAS

Invariants

- RepDay is the inverse of IntDay; RepTime is the inverse of IntTime.
- Any pair of day and time records from the OPAS points to a single time in the domain.
- Now is represented in the information system partly by as-NowD, and partly by as-NowT.
- If time t2 is Next after time t1, then either the representation of t1 precedes the representation of t2, or t1 is the last time record for one day record, t2 is the first time record for another day record, and the first day record precedes the second.

ClockInteraction.Tick()

dis-int-Pr 1: **DIS1Clock.Tick()**

dis-int-Pr 2: **Clock.Tick()**

Preconditions:

- Inherit the preconditions and postconditions of the Tick operation from the class DIS1Clock.
- Inherit the preconditions and postconditions of the Tick operation from the class Clock.

DIS1TypeInteraction

CRSTypeClass1, TypeClass5

dis-int-T 3: IntT: crs-Types → Types; RepT: Types ⇸ crs-Types

dis-int-T 4: SupTypes, SupBook, SupAccess, FullRepTypes: Set[Types]

dis-int-I 5: $IntT = RepT^{-1}$

dis-int-I 6: $IntT \circ crs\text{-}TypeParent \circ RepT \subseteq Cod(TypeGuide)$

dis-int-I 7: SupTypes = Dom(RepT); SupBook = (im IntT) (crs-Bookable); SupAccess = (im IntT) (crs-Accessible)

dis-int-I 8: $FullRepTypes \subseteq SupTypes \subseteq HomeTypes$

dis-int-I 9: (im RepT) FullRepTypes $\subseteq$ crs-Types \ Cod(crs-TypeParent)

dis-int-I 10: SupBook = SupType $\cap$ Bookable; SupAccess = SupType $\cap$ Access

dis-int-I 11: (im IntT) (crs-Types\Cod(crs-TypeParent)) $\subseteq$ PatReq

dis-int-I 12: $\forall t: SupTypes \bullet \sim\exists tg1, tg2: TGroupers \bullet tg1 \neq tg2 \wedge t \in Dom(TypeGuide(tg1) \cap TypeGuide(tg2))$

NB The state components that start with the prefix Sup- are all subsets of state components from the domain. In particular they are members of domain state components that are represented, or supported in the information system: thus SupTypes is the set of all members of Types that are represented in DIS1.

Types

- IntT is a function from types as represented in the CRS to types in the domain
- RepT is a partial function from types in the domain to their representation in the CRS
- SupTypes, SupBook and SupAccess, and FullRepTypes are all sets of types. FullRepTypes is the set of types all of whose activities are represented in the DIS. More specifically it is the set of types such

that the start of an activity of this type in the domain is always accompanied by the creation and 'start' of a corresponding activity record in the information system.

Invariants

- IntT is the inverse of RepT.
- The function TypeParent, interpreted into the domain, is a subset of the codomain of TypeGuide. There is no representation of TGrouper as multiple embedding of visit records is not allowed in the CRS or DIS1.
- SupTypes is the set of types that are suported by the DIS; SupBook is the interpretation of the set Bookable in the CRS; SupAccess is the interpretation of the set Accessible in the CRS.
- All members of FullRepTypes are supported types, and all supported types are home types.
- Each member of FullRepTypes is represented by a 'childless' type record in the information system. This is to ensure that a started activity of a type in FullRepTypes can always be represented. If we did not have this invariant then we could start a child activity unbeknownst to the information system which would start a parent of a type that we are trying to ensure is represented.
- SupBook is all supported types that are bookable; SupAccess is all supported types that are in Access.
- All types that have no children through crs-TypeParent are representations of types in PatReq.
- Any type in SupTypes only appears in one relation pointed at by a member of TGrouper. Thus blood tests and similar activities are not supported (in fact they are supported by the CRS, but in a completely different manner - as attributes of clinical activities and not as clinical activities in their own right.

DIS1Interaction

DIS1Class1, DIS1TypeInteraction, ClockInteraction, ATClass5

dis-int-T 5: IntA: dis-Activities → Activities; RepA: Activities ⇸ dis-Activities

dis-int-T 6: IntP: crs-Pid → Patients; RepP: Patients ⇸ crs-Pid

dis-int-T 7: RepSlot: Slots ⇸ as-Slots; IntSlot: as-Slots ⇸ Slots

dis-int-T 8: IntClist: dis-Clist → C; RepClist: C ⇸ dis-Clist

dis-int-T 9: SupAct: Set[Activities]; SupPat: Set[Patients]; SupSlots: Set[Slots]

dis-int-I 13: $IntA = RepA^{-1}$; $IntP = RepP^{-1}$; $IntSlot = RepSlot^{-1}$; $IntClist = RepClist^{-1}$

dis-int-I 14: SupAct = Dom(RepA); SupPat = Dom(RepP); SupSlot = Dom(RepSlot)

dis-int-I 15: $SupAct \subseteq (im\ ActType^{-1})\ SupTypes$

dis-int-I 16: IntA ⚬ dis-VisRel ⚬ RepA ⊆ During

dis-int-I 17: IntT ⚬ dis-ActType ⚬ RepA = SupAct ◁ ActType

dis-int-I 18: IntP ⚬ dis-ActSubject ⚬ RepA = SupAct ◁ ActSubject

dis-int-I 19: IntSlot ⚬ dis-ActSlot ⚬ RepA = SupAct ◁ ActSlot

dis-int-I 20: $(im\ ActType^{-1})\ FullRepTypes \cap Proceed \subseteq (im\ IntA)\ crs\text{-}Proceed$

dis-int-I 21: SupAct ∩ Proceed \ (im IntA) Cod(dis-VisRel) ⊆ (im IntA) crs-Proceed

dis-int-I 22: SupAct ∩ Complete \ (im IntA) Cod(dis-VisRel) = (im IntA) crs-Complete

dis-int-I 23: RepT ⚬ ((ActType ▷ ((im IntT) dis-Bookable)) ⚬ $dis\text{-}ActSlot^{-1}$ ⚬ $as\text{-}SlotClist^{-1}$) ⚬ IntClist ⊆ $dis\text{-}TypeLink^{-1}$ ⚬ (id[as-CTypes] ◊ as-CTypeModes) ⚬ (ClinicType ⚬ StreamClinic ⚬ StreamClist)

dis-int-I 24: Cod(RepSlot) = Dom(as-Appointments)

dis-int-I 25: RepDay ⚬ $(SupSlot \triangleleft SlotStart)^{-1}$ ⚬ (SupSlot ◁ SlotEnd) ⚬ IntDay ⊆ id[as-Days]

dis-int-I 26: as-SlotDay = RepDay ⚬ (SupSlot ◁ SlotStart) ⚬ IntSlot

dis-int-I 27: RepTime ⚬ (SupSlot ◁ SlotStart) ⚬ IntSlot = as-SlotStart

dis-int-I 28: RepTime ⚬ (SupSlot ◁ SlotEnd) ⚬ IntSlot = as-SlotEnd

dis-int-I 29: as-StartTime ⊆ RepTime ⚬ (SupSlot ◁ (ActStart ⚬ $ActSlot^{-1}$)) ⚬ IntSlot

dis-int-I 30: as-EndTime ⊆ RepTime ⚬ (SupSlot ◁ (ActEnd ⚬ $ActSlot^{-1}$)) ⚬ IntSlot

- Inherit the type declarations and invariants of the classes DIS1Class1, DIS1TypeInteraction, ClockInteraction, and ATClass5

Types:

- IntA is a total function from activity records in DIS1 to activities in the domain; RepA is a partial function from activities in the domain to activity records in the DIS.
- IntP is a total function from patient ids to patients; RepP is a partial function from patients to patient ids.
- RepSlot is a partial function from slots in the domain to slot records in the OPAS; IntSlot is a partial function from slot records in the OPAS to slots in the domain.
- IntClist is a total function from clinic list records in DIS1 to clinic lists in the domain; RepClist is a partial function from clinic lists in the domain to clinic list records in the DIS1.

- SupAct is a set of activities; SupPat is a set of patients; SupSlots is a set of slots.

Invariants

- IntA is the inverse of RepA; IntP is the inverse of RepP; IntSlot is the inverse of RepSlot; IntClist is the inverse of RepClist.
- SupAct is all activities represented in DIS1; SupPat is all patients represented in DIS1; SupSlot is all slots represented in DIS1.
- Any activity in SupAct is of a type in SupTypes.
- The projection of the tree dis-VisRel onto Activity space is a subset of During.
- The projection of dis-ActType onto domain space (ie state components from the domain theory) is that subset of ActType that has SupAct as its domain.
- The projection of dis-ActSubject onto domain space is that subset of ActSubject that has SupAct as its domain.
- The projection of dis-ActSlot onto domain space is that subset of dis-ActSlot that has SupAct as its domain.
- The start of all activities of a type in FullRepTypes is recorded in the information system.
- The start of all requests represented by a 'childless' activity record is recorded.
- The completion of all activities represented by a 'childless' activity record is recorded: the completion of no other activities is recorded.
- All complete supported activities are represented as members of the set Complete in the DIS
- The various types associated with a particular clinic list (through activities associated with the clinic list) are allowable types. We can see if they are allowable by looking at the dis-Clist record the particular clinic list is associated with (through IntClist) and seeing what types can be feasibly associated with it. What this invariant means is that if the activity is supported, then, if it is to be booked, it must be booked on the appointment system. This in turn imposes constraints on the domain, as we are not free to associate a clinic list with any type of activity on the information system.
- All slots that have patients associated with them in the OPAS represent slots in the domain through the function RepSlot.
- All supported slots begin and end on the same day
- The day record the representation of a slot is stated as starting on is interpreted as the day that the slot in the domain is booked to start on.
- The scheduled start of a slot in the OPAS can be translated to the start of the (interpreted) slot in the domain.
- The scheduled end of a slot in the OPAS can be translated to the end of the (interpreted) slot in the domain.
- The actual start of the slot in the OPAS can be translated as the start of the relevant activity in the domain.
- The actual end of the slot in the OPAS can be translated as the end of the relevant activity in the domain.

DIS1Interaction.NonRecCreate(A_b,p_n,t_n→a_n)

dis-int-Pr 3: **ATClass5.Create(A_b,p_n,t_n→a_n)**

Preconditions:

- Inherit the preconditions and postconditions of the the .Create operation from the ATClass5 class.

DIS1Interaction.RecCreate(A_b,p_n,t_n→a_n)

dis-int-Pr 4: **ATClass5.Create**(A_b,p_n,t_n→a_n)

dis-int-Pr 5: **DIS1Class1.Create**(p_n,RepT(t_n)→v_n)

---

dis-int-Po 1: (v_n,a_n) ∈ IntA'

Preconditions:

- Inherit the preconditions and postconditions of the .Create operation from the ATClass5 class.
- Inherit the preconditions and postconditions of the .Create operation from the DIS1Class1 class (note that one of the preconditions inherited by this operation tells us that RepT(t_n): crs-Types - thus t_n must be in SupTypes).

Postconditions:

- v_n (returned from the invoked information system operation) is now the representation of a_n.

DIS1Interaction.NonRecGenerate(A_p,A_b,t_c,hcp→a_c)

dis-int-Pr 6: **ATClass5.Generate**(A_p,A_b,t_c,hcp→a_c)

Preconditions:

- Inherit the preconditions and postconditions of the .Generate operation from the ATClass5 class.

DIS1Interaction.RecGenerate(A_p,A_b,t_c,hcp→a_c)

dis-int-Pr 7: p: (im ActSubject) A_p

dis-int-Pr 8: **DIS1Class1.Generate**(RepT(t_c),RepP(p)→v_c)

dis-int-Pr 9: **ATClass5.Generate**(A_p,A_b,t_c,hcp→a_c)

dis-int-Pr 10: IntT ⚬ ((Cod({RepT(ActType(a))} ◁ crs-TypeParent*)) ◁ crs-TypeParent) ⚬ RepT ⊆ ActType ⚬ ((Cod({a} ◁ During*)) ◁ During) ⚬ ActType^{-1}

---

dis-int-Po 2: (v_c,a) ∈ IntA'

Preconditions:

- p is one of the subjects of the set of 'parent' activities A_p (the set of such subjects is specified as a singleton set in one of the invariants of the domain theory).
- Inherit the preconditions and postconditions of the .Generate operation from the DIS1Class1 class.
- Inherit the preconditions and postconditions of the .Generate operation from the ATClass5 class.
- The structure that is created as a result of calling .Generate from the DIS1Class1 class must be capable of being matched up with an existing structure in the domain that has a as its most junior member.

Postconditions:

- a is now the interpretation of v_c.

DIS1Interaction.NonRecSuddenStart(A_p,t_c,hcp→a_c)

dis-int-Pr 11: **ATClass5.SuddenStart**(A_p,t_c,hcp→a_c)

dis-int-Pr 12: $t_c \notin$ FullRepTypes

Preconditions:

- Inherit the preconditions and postconditions of the the .Create operation from the ATClass5 class.
- t_c is not a member of FullRepTypes.

DIS1Interaction.RecSuddenStart(A_p,t_c,hcp→a_c)

dis-int-Pr 13: p: (im ActSubject) A_p

dis-int-Pr 14: **ATClass5.SuddenStart**(A_p,t_c,hcp→a_c)

dis-int-Pr 15: **DIS1Class1.SuddenStart**(RepT(t_c),RepP(p)→v_c)

dis-int-Pr 16: IntT ° ((Cod({RepT(ActType(a))} ◁ crs-TypeParent*)) ◁ crs-TypeParent) ° RepT ⊆ ActType ° ((Cod({a} ◁ During*)) ◁ During) ° ActType^{-1}

dis-int-Po 3: (v_c,a) ∈ IntA'

Preconditions:

- p is one of the subjects of the set of 'parent' activities A_p.
- Inherit the preconditions and postconditions of the SuddenStart operation from the ATClass5 class.
- Inherit the preconditions and postconditions of the SuddenStart operation from the DIS1Class1 class.
- The structure that is created as a result of calling .Generate from the DIS1Class1 class must be capable of being matched up with an existing structure in the domain that has a as its most junior member.

Postconditions:

- a is now the interpretation of v_c.

DIS1Interaction.Book(A_p,A_b,t_c,hcp,c,τ_b,τ_e→a_c,s)

dis-int-Pr 17: **ATClass5.Book**(A_p,A_b,t_c,hcp,c,τ_b,τ_e→a_c,s)

dis-int-Pr 18: p: (im ActSubject) A_p

. 1

dis-int-Pr 19: $t_c \notin$ (im IntT) dis-Bookable

. 2

dis-int-Pr 20: $t_c \in$ (im IntT) dis-Bookable

dis-int-Pr 21: $v_p \in$ (im RepA) $A_p \cap$ (im dis-ActType^{-1}) {TypeParent(tc)}

dis-int-Pr 22: c ∈ Dom(RepClist) ⇒ dis-c = RepClist(c)

dis-int-Pr 23: c ∉ Dom(RepClist) ⇒ dis-c ∈ dis-Cl\Cod(RepClist)

dis-int-Pr 24: c ∈ Dom(RepClist) ⇒ $t_c \in$ (im dis-TypeLink^{-1} ° (id[as-CTypes] ◊ as-CTypeModes) ° (ClinicType ° StreamClinic ° StreamClist)) {dis-c}

dis-int-Pr 25: RepDay(τ_b) = RepDay(τ_e)

dis-int-Pr 26: ∃sl: dis-Slots • as-SlotStart(sl) = RepTime(τ_b) ∧ as-SlotEnd(sl) = RepTime(τ_e) ∧ dis-TypeLink(RepT(t)) = ((as-ClinicType ° as-StreamClinic ° as-SlotStream) ◊ as-SlotMode) (sl)

dis-int-Pr 27: IntT ° ((Cod({RepT(ActType(a))} ◁ crs-TypeParent*)) ◁ crs-TypeParent) ° RepT ⊆ ActType ° ((Cod({a} ◁ During*)) ◁ During) ° ActType^{-1}

dis-int-Pr 28: **DIS1Class1.Book**(RepT(t_c),RepP(p),dis-c,RepDay(τ_b),RepTime(τ_b)→v_c,s)

---

dis-int-Po 4: (c,dis-c) ∈ RepClist'

dis-int-Po 5: $t_c \in$ (im dis-TypeLink^{-1} ° (id[as-CTypes] ◊ as-CTypeModes) ° (ClinicType ° StreamClinic ° StreamClist')) {dis-c}

dis-int-Po 6: (v_c,a) ∈ IntA'

Preconditions:

- Inherit the preconditions and postconditions of the .Book operation from the ATClass5 class.
- p is one of the subjects of the set of 'parent' activities A_p.

  Case 1:

  Preconditions:

  - t_c is not represented by a type in dis-Bookable.

  Case 2:

  Preconditions:

  - t_c is represented by a type in dis-Bookable (This condition is the one that insists that all supported activity types that are booked must be booked through the information system).
  - v_p is the representation of one of the activities in A_p - the one that is of a type that is the parent of the representation of t_c.
  - If the clinic list c is represented in the DIS then dis-c is its representation
  - If the clinic list c is not represented in the DIS then dis-c is of the same type as representations of clinic lists, but is not itself such a representation

- t_c is the interpretation of one of the allowable types that can be associated with the 'list' dis-c. If dis-c is known (c is represented in the DIS), then we can work out whether t_c is an allowable type by looking at: the clinic type associated with all clinics associated with all streams associated with the list; and all allowable modes associated with the clinic type. From the clinic type and mode, the type can be calculated through TypeLink.
- The slot starts on the same day as it ends
- Thre is an available slot in the OPAS that starts at the right time, ends at the right time, and is associated with clinics of an allowable typc.
- The structure that is created as a result of calling .Book from the DIS1Class1 class must be capable of being matched up with an existing structure in the domain that has a as its most junior member.
- Inherit the preconditions and postconditions of the .Book operation from the DIS1Class1 class.

Postconditions

- dis-c is now the representation of c.
- tc is an allowable type to be associated with dis-c (which is important if we didn't know what c was before the operation).
- a is now the interpretation of v_c.

DIS1Interaction.Schedule(c,τ_b,τ_e,a,s)

dis-int-Pr 29: **ATClass5.Book**(c,τ_b,τ_e,a,s)

. 1 dis-int-Pr 30: $a_c \notin$ SupAct

. 2

dis-int-Pr 31: $a \in$ SupAct

dis-int-Pr 32: $c \in$ Dom(RepClist) $\Rightarrow$ dis-c = RepClist(c)

dis-int-Pr 33: $c \notin$ Dom(RepClist) $\Rightarrow$ dis-c $\in$ dis-Clist\Cod(RepClist)

dis-int-Pr 34: $c \in$ Dom(RepClist) $\Rightarrow$ ActType(a) $\in$ (im dis-TypeLink^{-1} $\circ$ (id[as-CTypes] $\Diamond$ as-CTypeModes) $\circ$ (ClinicType $\circ$ StreamClinic $\circ$ StreamClist)) {dis-c}

dis-int-Pr 35: RepDay(τ_b) = RepDay(τ_e)

dis-int-Pr 36: $\exists$sl: dis-Slots • as-SlotStart(sl) = RepTime(τ_b) $\wedge$ as-SlotEnd(sl) = RepTime(τ_e) $\wedge$ dis-TypeLink(RepT(t)) = ((as-ClinicType $\circ$ as-StreamClinic $\circ$ as-SlotStream) $\Diamond$ as-SlotMode) (sl)

dis-int-Pr 37: **DIS1Class1.Schedule**(RepA(a),dis-c,RepDay(τ_b),RepTime(τ_b)$\rightarrow$s)

dis-int-Po 7: (c,dis-c) $\in$ RepClist'

dis-int-Po 8: ActType(a) $\in$ (im dis-TypeLink^{-1} $\circ$ (id[as-CTypes] $\Diamond$ as-CTypeModes) $\circ$ (ClinicType $\circ$ StreamClinic $\circ$ StreamClist)) {dis-c}

Preconditions:

- Inherit the preconditions and postconditions of the .Book operation from the ATClass5 class.

  Case 1:

  Preconditions:

  - a_c is not a supported activity.

Case 2:

Preconditions:

- t_c is a supported type.
- If the clinic list c is represented in the DIS then dis-c is its representation.
- If the clinic list c is not represented in the DIS then dis-c is of the same type as representations of clinic lists, but is not itself such a representation.
- The type of a is the interpretation of one of the allowable types that can be associated with the clinic list record dis-c.
- The slot starts on the same day as it ends
- There is an available slot in the OPAS that starts at the right time, ends at the right time, and is associated with clinics of an allowable type.
- Inherit the preconditions and postconditions of the .Book operation from the DIS1Class1 class, with dis-c as the clinic list record.

Postconditions

- dis-c is now the representation of c
- The type of a is an allowable type to be associated with dis-c (which is important if we didn't know what c was before the operation)

DIS1Interaction.NonRecStart(a)

dis-int-Pr 38: **ATClass5.Start**(a)

. 1 dis-int-Pr 39: ActType(a) ∉ FullRepTypes

dis-int-Pr 40: a ∉ SupAct \ (im IntA) Cod(dis-VisRel)

. 2 dis-int-Pr 41: Rep(a) ∈ crs-Proceed

Preconditions:

- Inherit the preconditions and postconditions of the .Start operation from the ATClass5 class.

  Case 1:

  Preconditions:

  - a is not of a type in FullRepTypes
  - The only time a can be a supported activity is when it is 'parent' activity record (as defined through dis-VisRel).

  Case 2:

  - a is represented by an activity record in crs-Proceed. This represents the case when the activity has been suspended in the domain and restarted - a domain operation that is not supported by the information system.

DIS1Interaction.RecStart(a)

dis-int-Pr 42: **ATClass5.Start**(a)

dis-int-Pr 43: ActType(a) ∈ SupTypes

. 1
dis-int-Pr 44: a ∉ SupAct

dis-int-Pr 45: IntT ° ((Cod({RepT(ActType(a))} ◁ crs-TypeParent*)) ◁ crs-TypeParent) ° RepT ⊆ ActType ° ((Cod({a} ◁ During*)) ◁ During) ° ActType^{-1}

dis-int-Pr 46: **DIS1Class1.SuddenStart**(RepT(ActType(a)),RepP(ActSubject(a))→v_s)

dis-int-Po 9: $(v_s,a) \in IntA'$

. 2
dis-int-Pr 47: a ∈ SupAct

dis-int-Pr 48: **DIS1Class1.Start**(RepA(a))

Preconditions:

- Inherit the preconditions and postconditions of the .Start operation from the ATClass5 class.
- a is of a type supported by the information system.

  Case 1:

  Preconditions:

  - a is not currently represented in the information system.
  - The structure that is created as a result of calling .SuddenStart from the DIS1Class1 class must be capable of being matched up with an existing structure in the domain that has a as its most junior member.
  - Inherit the preconditions and postconditions of the .SuddenStart operation from the DIS1Class1 class.

  Postconditions:

  - a is now the interpretation of v_s.

  Case 2:

  - a is an activity represented in the information system.
  - Inherit the preconditions and postconditions of the .Start operation from the DIS1Class1 class.

DIS1Interaction.Suspend(a)

dis-int-Pr 49: **ATClass5.Suspend**(a)

NB This operation is not represented in the IS

DIS1Interaction.End(a)

dis-int-Pr 50: a ∈ SupAct ⇒ **DIS1Class1.End**(RepA(a))

dis-int-Pr 51: **ATClass5.End**(a))

Preconditions:

- If a is represented in the information system then inherit the preconditions and postconditions of the .End operation from the DIS1Class1 class.
- Inherit the preconditions and postconditions of the .End operation from the ATClass5 class.

DIS1Interaction.Cancel(a)

dis-int-Pr 52: a ∈ SupAct ⇒ **DIS1Class1.Cancel**(RepA(a))

dis-int-Pr 53: **ATClass5.Cancel**(a)

Preconditions:

- If a is represented in the information system then inherit the preconditions and postconditions of the .Cancel operation from the DIS1Class1 class.
- Inherit the preconditions and postconditions of the .Cancel operation from the ATClass5 class.

DIS1Interactionion.PatReg(p)

dis-int-Pr 54: **ATClass5.PatReg**(p)

NB This operation is not explicitly represented in the IS

DIS1Interaction.PatArrive(p)

dis-int-Pr 55: **ATClass5.PatArrive**(p)

NB This operation is not represented in the IS

DIS1Interaction.NotePatArrive(pid,τ)

dis-int-Pr 56: **DIS1Class1.PatArrive**(pid,τ)

dis-int-Pr 57: IntP(pid) ∈ PatPres

Preconditions:

- Inherit the preconditions and postconditions of the PatArrive operation from the DIS1Class1 class.
- pid is the patient id of a patient who is currently present

DIS1Interaction.PatDepart(p)

dis-int-Pr 58: **ATClass5.PatDepart**(p)

NB This operation is not represented in the IS

DIS1Interaction.OutCreate(p_n,t_n,hcp→a_n)

dis-int-Pr 59: **ATClass5.OutCreate**(p_n,t_n,hcp→a_n)

NB This operation is not represented in the IS

DIS1Interaction.OutGenerate(A_p,t_c,hcp→a_c)

dis-int-Pr 60: **ATClass5.OutGenerate**(A_p,t_c,hcp→a_c)

NB This operation is not represented in the IS

DIS1Interaction.Associate(a,hcp)

dis-int-Pr 61: **ATClass5.Associate**(a,hcp)

NB This operation is not represented in the IS

DIS1Interaction.Disassociate(hcp)

dis-int-Pr 62: **ATClass5.Disassociate**(hcp)

NB This operation is not represented in the IS

DIS1Interaction.OutProceed(a)

dis-int-Pr 63: **ATClass5.OutProceed**(a)

NB This operation is not represented in the IS

DIS1Interaction.OutComplete(a)

dis-int-Pr 64: **ATClass5.OutComplete**(a)

NB This operation is not represented in the IS

DIS1Interaction.PatJoin(a,p)

dis-int-Pr 65: **ATClass5.PatJoin**(a,p)

NB This operation is not represented in the IS

DIS1Interaction.PatLeave(p)

dis-int-Pr 66: **ATClass5.PatLeave**(p)

NB This operation is not represented in the IS

DIS1Interaction.ClinicSetup(ct,D,Dn,DClist→Cl)

dis-int-Pr 67: **DIS1Class1.ClinicSetup**(ct,D,Dn→Cl)

NB Not representative of an operation in the domain

DIS1Interaction.StreamCancel(st)

dis-int-Pr 68: **DIS1Class1.StreamCancel**(st)

NB Not representative of an operation in the domain

DIS1Interaction.SlotsCreate(TAm,st)

dis-int-Pr 69: **DIS1Class1.SlotsCreate**(TAm,st)

NB Not representative of an operation in the domain

DIS1Interaction.Tick()

dis-int-Pr 70: **ClockInteraction.Tick**()

Preconditions:

- Inherit the preconditions and postconditions of the .Tick operation from the ClockInteraction class.

Appendix 6:

The Extension of the Domain Theory to a Hypothetical Domain: Out-patient Contracting

This appendix presents a refinement of ATClass5 of the domain theory. The new theory describes the behaviour of the directorate as it might be following the introduction of more efficient contracting procedures. In order to represent the new organisation a number of new state components have been introduced. However, mutual constraints mean that the ratio of forbidden to permitted behaviours is increased so the theory is an emboldening of the original.

The envisaged behaviour was recorded after discussions with the directorate manager (Mrs Kay Checkley) and the hospital contracts manager (Mr Ray Franklin). A more detailed description of the analysis process is recorded in [Sangh94].

PackageTypeClass

TypeClass5

PType, PTClass_Time, PTClass_No: Set[pT]

PT_Duration: PType ⇸ N+

PTypeAType: PType ↔ Type

PTATNo: PTypeAType ⇸ N+

PTClass_Time ∪ PTClass_No = PType; PTClass_Time ∩ PTClass_No = ∅

Dom(PT_Duration) = PTClass_Time

Cod(PTypeAType) ⊆ Bookable

Dom(Dom(PTATNo)) = PTClass_No

Declarations

- PType (a set of package types), PTClass_Time (the subset of types that deliver activities for a fixed time), PTClass_No (the subset of types that deliver a fixed number of activities) are all sets of the same type.
- PT_Duration is a function that returns a positive number (the time duration of packages of this type) when supplied with a package type.
- PTypeAType is a relation that associates package types with activity types
- PTATNo is a function that returns a positive number (the number of activities of the activity type that can be associated with a package of the package type) when supplied with an activity type - package type pair.

Invariants

- PTClass_Time and PTClass_No completely partition he set and are disjoint.
- All package types that have a duration are in the set PTClass_Time.
- All activity types that are associated with package types are elements of the set Bookable - the set of bookable activity types.
- All package types that appear in the 'triple' PTATNo are in the set PTClass_No.

Examples of the sets PTypeAType and PTATNo are given below:

PTypeAType:

Diabetes Package (Year)	Init Dr Cons
Diabetes Package (Year)	Followup Dr Cons
First Diabetes Package (Activity)	Init Dr Cons
First Diabetes Package (Activity)	Followup Dr Cons
Subsequent Diabetes Package (Activity)	Followup Dr Cons
First Single Visit Package	Init Dr Cons
Subsequent Single Visit Activity	Followup Dr Cons

PTATNo:

First Diabetes Package (Activity)	Init Dr Cons	1
First Diabetes Package (Activity)	Followup Dr Cons	2
Subsequent Diabetes Package (Activity)	Followup Dr Cons	3
First Single Visit Package	Init Dr Cons	1
Subsequent Single Visit Activity	Followup Dr Cons	1

PackageClass1

PackageTypeClass

Packages, Pack_Secured, Pack_Pending, Pack_Refused, Pack_Delivered, Pack_Terminated, Pack_Complete: Set[Pk]

PackType: Packages → PType

ActPack: Activities ⇸ Packages

PackPat: Packages → Patients

PackStart: Packages ⇸ T

PatPType: Patients ⇿ PType

PackAType: Packages ↔ Types

PatPkAType: Patients ⇿ Types

ActPackAType: Activities ⇸ (Pack X Types)

Pack_Secured ∪ Pack_Pending ∪ Pack_Refused = Packages

Pack_Secured ∩ Pack_Pending = Pack_Pending ∩ Pack_Refused = Pack_Refused ∩ Pack_Secured = ∅

Pack_Delivered ∪ Pack_Terminated = Pack_Complete; Pack_Delivered ∩ Pack_Terminated = ∅

PackPat ⁰ ActPack ⊆ ActSubject

Dom(ActPack ⩥ Pack_Complete) ⊆ Complete

PatPType = PackType ⁰ PackPat^{-1}; PackAType = PTypeAType ⁰ PackType

PatPkAType = PackAType ⁰ PackPat^{-1}; ActPackAType = ActPack ◊ ActType

∀pk: Dom(PackType ⩥ PTClass_No) • ∀t: (im PackAType) {pk} • #(im ActPackAType^{-1}) {(pk,t)} ≤ PTATNo((PackType(p),t))

∀pk: Dom(PackType ⩥ PTClass_No) • pk ∈ Pack_Delivered ⇔ ∀t: (im PackAType) {pk} • #(im ActPackAType^{-1}) {(pk,t)}\Request = PTATNo((PackType(p),t))

∀p: Patients • ∀t: (im PatPkAType){p} •
∀pk_1,pk_2: (im PackPat^{-1}){p} ∩ (im PackAType^{-1}) {t} \ Pack_Complete •
(PackType(pk_1) ∈ PTClass_No ∧ #(im ActPackAType^{-1}) {(pk_1,t)} = PTATNo((PackType(pk_1),t))) ∨
(PackType(pk_2) ∈ PTClass_No ∧ #(im ActPackAType^{-1}) {(pk_2,t)} = PTATNo((PackType(pk_2),t))) ∨ pk_1 = pk_2

Packages' = ∅

Declarations

- Packages, Pack_Secured (the set of packages for which contracts have been secured), Pack_Pending (the set of packages for which a contract is being awaited), Pack_Refused (the set of packages for which a contract has been refused by the purchaser), Pack_Delivered (the set of packages successfully and completely delivered), Pack_Terminated (the set of packages that have been abnormally terminated), and Pack_Complete (the set of packages that have bee completed normally or abnormally) are sets of the same type.
- PackType records the package type of every package
- ActPack records the package (if any) that an activity has contributed towards the completion of.
- PackPat records the patient that a package has been created for

- *PackStart records the time of commencement of all packages that have started*
- PatPType is a relation that records the package types of the packages associated with a particular patient
- PackAType is a relation from Packages to activity types. For a particular package, only visits of types listed in the image of that package through the relation can be associated with the package
- PatPkAType is a relation from patients to activity types. The image of a patient through this relation records the allowable types that can be associated with packages associated with the patient
- ActPackAType is a function from activities to a pair of packages and activity types. When supplied with an activity, the function returns the package and activity type associated with that activity.

Invariants

- Pack_Secured, Pack_Pending and Pack_Refused partition the set of packages
- Pack_Secured, Pack_Pending and Pack_Refused are disjoint
- Pack_Complete is the union of Pack_Delivered and Pack_Terminated
- Pack_Delivered and Pack_Refused are disjoint
- The patient that a package is associated with is the same as the-patient that all that package's activities are associated with
- All activities that a completed (normally or abnormally) package is associated with are themselves complete.
- PatPType gives the set of types of packages associated with a patient
- PackAType gives the allowable activity types that can be associated with a package
- PatPkAType gives the allowable activity types that can be associated with all packages associated with a patient
- ActPackAType returns the package identifier and activity type associated with an activity
- For any package that is of a package type in PTClass_No, for any activity type that might describe activities feasibly associated with that package, the number of activities of that type that are associated with the package must be less than the maximum allowed number as defined by PTATNo
- For any package that is of a package type in PTClass_No, if the package is delivered, then for any activity type that might describe activities feasibly associated with that package, the number of activities of that type that are associated with the package must be equal to the maximum allowed number as defined by PTATNo, and *vice versa*
- For all patients, for all allowable activity types that can be associated with packages associated with that patient, suppose there are two distinct, non-complete, packages which support activities of that type, and are assigned to that patient: one of the packages must be complete with respect to that type. We know if a package is complete with respect to a type if it is of a package type in PTClass_No and the number of activities of the type in question associated with the package is the maximum number as defined by PTATNo. A package cannot be complete wrt a particular activity type if it is of a package type that is in PTClass_Time

PackageClass1.PackCreate(pt,p → pk)

pt: PType; p: Patients; pk: Pk\Packages
∀t: (im PTypeAType) {pt} • ~∃pk_0: (im $PackPat^{-1}$){p} ∩ (im $PackAType^{-1}$) {t} \ Pack_Complete • (PackType(pk_0) ∈ PTClass_Time) ∨ (PackType(pk_0) ∈ PTClass_No ∧ #(im $ActPackAType^{-1}$) {(pk_0,t)} < PTATNo((PackType(pk_0),t)))
pk ∈ Pack_Pending' ∪ Pack_Secured'
(pk,pt) ∈ PackType'
(pk,p) ∈ PackPat'

Types

- pt is a package type
- p is a (registered) patient
- pk is of the same type as Packages but is not a package

Preconditions

- For all the activity types that can be associated with the type of the new package, there is no non-complete package that is associated with the patient and can support activities of the type in question where that package has a package type in PTClass_Time or where that package is not complete wrt that activity type

Postconditions

- The new package, pk, is either in Pack_Pending or Pack_Secured
- The new package is now of package type pt.
- The new package is now associated with the patient p

PackageClass1.PackActAss(pk,p,t→a)

pk: Packages \ (Pack_Delivered ∪ Pack_Terminated)
p: Patients; t: Bookable; a: A
p = PackPat(pk)
t ∈ (im PTypeAType) {PackType(pk)}
PackType(pk) ∈ PTClass_No ⇒ #((im $ActPack^{-1}$) {p}) ◁ ActType ▷ {t} < PTATNo((PackType(p),t))
(a,pk) ∈ ActPack'

Types

- pk is a package that has not been completed
- p is a patient
- t is a bookable activity type
- a is of the same type as Activities, but might not be one yet

Preconditions

- p is the patient associated with the package pk
- t is a type that is associated with the package type through PTypeAType

- If the package is of a package type in PTClass_No then the number of activities associated with pk that are of type t is less than the maximum that can be associated before the package is complete

Postconditions

- pk is now associated with a through the function ActPack. a must now be in Activities (it cannot be in A\Activities) as the range of ActPack is a subset of Activities.

PackageClass1.PackSecure(pk)

pk: Pack_Pending
pk ∈ Pack_Secured'

Types

pk is a package in Pack_Pending

Postconditions

pk is now in Pack_Secured

PackageClass1.PackStart(pk,τ)

pk: Packages \ (Pack_Delivered ∪ Pack_Terminated) τ: T pk ∉ Dom(PackStart)
(pk,τ) ∈ PackStart'

Types

- pk is an incomplete package
- τ is a time

Preconditions

- the package pk has not started

Postconditions

- The start time of pk is now τ.

PackageClass1.PackTerminate(pk)

pk: Packages \ (Pack_Delivered ∪ Pack_Terminated)
pk ∈ Pack_Terminated'

Types

- pk is an incomplete package

Postconditions

- pk is now in Pack_Terminated

PackageClass1.PackRefuse(pk)

pk: Pack_Pending
pk ∈ Pack_Refused'

Types

- pk is a pending package

Postconditions

- pk is now refused

PackageClass1.PackEnd(pk)

pk: Packages \ (Pack_Delivered ∪ Pack_Terminated)
pk ∈ Pack_Delivered'

Types

- pk is an incomplete package

Postconditions

- pk is now a delivered package

Other operations, inherited unchanged from ATClass5.

- **PackageClass1.Create**(A_b,p_n,t_n,a_n)
- **PackageClass1.InGenerate**(A_p,A_b,t_c,hcp,a_c)
- **PackageClass1.Book**(A_p,A_b,t_c,hcp,c,τ_b,τ_e,a_c,s)
- **PackageClass1.Book**(c,τ_b,τ_e,a_c,s)
- **PackageClass1.Unbook**(a)
- **PackageClass1.SuddenStart**(A_p,t_c,hcp,a_c)
- **PackageClass1.Start**(a)
- **PackageClass1.Suspend**(a)
- **PackageClass1.End**(a)
- **PackageClass1.Cancel**(a)
- **PackageClass1.Associate**(a,hcp)
- **PackageClass1.Disassociate**(hcp)
- **PackageClass1.OutCreate**(p_n,t_n,hcp,a_n)
- **PackageClass1.OutGenerate**(A_p,t_c,hcp,a_c)
- **PackageClass1.OutProceed**(a)
- **PackageClass1.OutComplete**(a)
- **PackageClass1.PatReg**(p)
- **PackageClass1.PatDereg**(p)
- **PackageClass1.PatArrive**(p)

- **PackageClass1.PatDepart**(p)
- **PackageClass1.PatJoin**(a,p)
- **PackageClass1.PatLeave**(p)

ContractConfig

DHA, ContDHA: Set[DHA]

GP, GPFH, GPFHBC: Set[GP]

ContDHA ⊆ DHA

GPFHBC ⊆ GPFH ⊆ GP

Declarations

- DHA (District Health Authorities) and ContDHA (DHAs with which the centre has a contract) are both sets of the same type (DHA)
- GP (General Practitioners), GPFH (GP Fund-Holders), and GPFHBC (GP Fund-Holders with whom the centre has a block contract) are all sets of the same type (GP)

Invariants

- ContDHA is a subset of DHA
- GPFHBC is a subset of GPFH which is a subset of GP

PackageClass2

PackageClass1, ContractConfig

Cont_ECR, Cont_GPFH, Cont_BC, Cont_GPFHBC: Set[Packages]

PackDHA: Packages → DHA; PackGP: Packages → GP

PatDHA: Patients → DHA; PatGP: Patients → GP

∀A,B: {Cont_ECR, Cont_GPFH, Cont_BC, Cont_GPFHBC} • A ∩ B = ∅ ∨ A = B

∪ {Cont_ECR, Cont_GPFH, Cont_BC, Cont_GPFHBC} = Packages

Cont_GPFH = Dom(PackGP ▷ GPFH \ GPFHBC)

Cont_GPFHBC = Dom(PackGP ▷ GPFHBC)

Cont_BC = Dom(PackDHA ▷ ContDHA) \ (Cont_GPFH ∪ Cont_GPFHBC)

Cont_ECR = Packages \ (Cont_GPFH ∪ Cont_BC ∪ Cont_GPFHBC)

Cont_GPFH ∪ Cont_BC ∪ Cont_GPFHBC ⊆ Pack_Secured

Declarations

- Cont_ECR (the set of packages that are extra-contractual), Cont_GPFH (the set of packages that are authorised by GP Fund-Holders that we do not have a block contract with), Cont_BC (the set of packages that are authorised by DHAs with whom we have a block contract), Cont_GPFHBC (the set of packages that are authorised by GP Fund-Holders that we have a block contract with) are all sets of packages.

- PackDHA returns the health authority that a supplied package is associated with
- PackGP returns the GP that a supplied package is associated with
- PatDHA returns the health authority that a supplied patient is associated with
- PatGP returns the GP that a supplied patient is associated with

Invariants

- The sets Cont_ECR, Cont_GPFH, Cont_BC, Cont_GPFHBC are disjoint
- The sets Cont_ECR, Cont_GPFH, Cont_BC, Cont_GPFHBC partition Packages
- Cont_GPFH is the set of packages that are associated with GP Fund-Holders that we do not have block contracts with
- Cont_GPFHBC is the set of packages that are associated with GP Fund-Holders that we have block contracts with
- Cont_BC is the set of packages that are associated with DHAs with whom we have a block contract, and not associated with GPs that are fundholders
- Cont_ECR is all packages that are not in Cont_GPFH, Cont_GPFHBC or Cont_BC.
- All packages in Cont_GPFH, Cont_BC or Cont_GPFHBC are automatically secured

PackageClass2.Register(gp,dha→p)

gp: GP; dha: DHA **PackageClass1.Register**(→p)
(p,gp) ∈ PatGP'; (p,dha) ∈ PatDHA'

Types

- gp is a GP
- dha is a DHA

Preconditions

- Invoke **PackageClass1.Register**

Postconditions

- gp is now the patient's assigned GP
- dha is now the patient's assigned DHA

PackageClass2.ChangeDemog(p,gp,dha)

p: Patients; gp: GP; dha: DHA
(p,gp) ∈ PatGP'; (p,dha) ∈ PatDHA'

Types

- p is a registered patient
- gp is a GP
- dha is a DHA

Postconditions

- gp is now the patient's assigned GP

- dha is now the patient's assigned DHA

PackageClass2.PackCreate(pt,p → pk)

PackageClass1.PackCreate(pt,p → pk)

. 1 PatGP(p) ∈ GPFHBC

pk ∈ Cont_GPFHBC' ∩ Pack_Secured'

. 2 PatGP(p) ∈ GPFH \ GPFHBC

pk ∈ Cont_GPFH' ∩ Pack_Secured'

. 3 PatGP(p) ∉ GPFH ∧ PatDHA(p) ∈ ContDHA

pk ∈ Cont_BC' ∩ Pack_Secured'

. 4 PatGP(p) ∉ GPFH ∧ PatDHA(p) ∉ ContDHA

pk ∈ Cont_ECR' ∩ Pack_Pending'

(pk,PatGP(gp)) ∈ PackGP'

(pk,PatDHA(dha)) ∈ PackDHA'

Preconditions

- invoke **PackageClass1**.PackCreate

Case 1

Preconditions

- The patient's current GP is a GP Fund-Holder with whom we have a block contract

Postconditions

- the new package is in Cont_GPFHBC and is secured

Case 2

Preconditions

- The patient's current GP is a GP Fund-Holder with whom we do not have a block contract

Postconditions

- the new package is in Cont_GPFH and is secured

Case 3

Preconditions

- The patient's GP is not a Fund-Holder and we have a contract with their DHA

Postconditions

- the new package is in Cont_BC and is secured

Case 4

Preconditions

- The patient's GP is not a Fund-Holder and we do not have a contract with their DHA

Postconditions

- The new package is in Cont_ECR and is pending contract secural

Postconditions

- The GP now associated with the package is the same as that associated with the patient
- The DHA now associated with the package is the same as that associated with the patient

PackageClass2.Create(A_b,p_n,t_n,a_n)

PackageClass1.Create(A_b,p_n,t_n,a_n)

. 1 pk: Packages

$t_n \in$ Cod(PTypeAType)

PackageClass1.PackActAss(pk,p_n,$t_n \rightarrow a_n$)

. 2 $t_n \notin$ Cod(PTypeAType)

Preconditions

- Invoke **PackageClass1.Create**

Case 1

Types

- pk is a package

Preconditions

- t_n is an activity type which is associated with a package type
- Invoke **PackageClass1.PackActAss**

Case 2

- t_n is not an activity type which is associated with a package type

PackageClass2.InGenerate(A_p,A_b,t_c,hcp,a_c)

PackageClass1.InGenerate(A_p,A_b,t_c,hcp,a_c)

. 1 pk: Packages

$t_n \in$ Cod(PTypeAType)

PackageClass1.PackActAss(pk,p_n,$t_n \rightarrow a_n$)

. 2 $t_n \notin$ Cod(PTypeAType)

Preconditions

- Invoke **PackageClass1.InGenerate**

Case 1

Types

- pk is a package

Preconditions

- t_n is an activity type which is associated with a package type
- Invoke **PackageClass1.PackActAss**

Case 2

- t_n is not an activity type which is associated with a package type

PackageClass2.Book(A_p,A_b,t_c,hcp,c,τ_b,τ_e,a_c,s)

PackageClass1.Book(A_p,A_b,t_c,hcp,c,τ_b,τ_e,a_c,s)

. 1 pk: Packages

$t_n \in$ Cod(PTypeAType)

PackageClass1.PackActAss(pk,p_n,$t_n \rightarrow a_n$)

. 2 $t_n \notin$ Cod(PTypeAType)

Preconditions

- Invoke **PackageClass1.Book** (The version which generates a new activity)

Case 1

Types

- pk is a package

Preconditions

- t_n is an activity type which is associated with a package type
- Invoke **PackageClass1.PackActAss**

Case 2

- t_n is not an activity type which is associated with a package type

PackageClass2.GoodStart(a)

a ∈ Dom(ActPack) ∧ ActPack(a) ∉ Dom(ActStart) ⇒ **PackageClass1.PackStart**(ActPack(a),now)

a ∈ Dom(ActPack) ⇒ ActPack(a) ∈ Pack_Secured

PackageClass1.Start(a)

. 1 pk: Packages; t: Types

a ∈ Dom(ActPack)

pk = ActPack(a); t = ActType(a)

PackType(pk) ∈ PTClass_No

PTATNo((PackType(pk),t)) - #((im $ActPack^{-1}$) {pk}) ∩ (im $ActType^{-1}$) {t} = 1

$\forall t_{nk}$: (im PTypeAType) {PackType(ActPack(a))} • PTATNo((PackType(pk),t_{nk})) = #((im $ActPack^{-1}$) {pk}) ∩ (im $ActType^{-1}$) {t} ∨ t_{pk} = t

PackageClass1.PackEnd(pk)

Preconditions

- If the activity to be started is associated with a package, and that package has not yet started then invoke **PackageClass1.PackStart**
- If the activity to be started is associated with a package then that package must be secured
- Invoke **PackageClass1.Start**

Case 1

Types

- pk is a package, t is an activity type

Preconditions

- The activity to be started is associated with a package
- pk is the package the activity is associated with
- t is the activity type of the activity
- The package type of pk is in PTClass_No
- The package, pk, is one activity short of completion with respect to t
- The package is already complete with respect to all (appropriate) types other than t
- Invoke **PackageClass1.PackEnd**

PackageClass2.BadStart(a)

pk: Packages; t: Types

a ∈ Dom(ActPack)

pk = ActPack(a); t = ActType(a)

pk ∉ Pack_Secured

ActPack(a) ∉ Dom(ActStart) ⇒ **PackageClass1.PackStart**(ActPack(a),now)

PackageClass1.Start(a)

. 1 PackType(pk) ∈ PTClass_No

PTATNo((PackType(pk),t)) - #((im $ActPack^{-1}$) {pk}) ∩ (im $ActType^{-1}$) {t} = 1

$\forall t_{nk}$: (im PTypeAType) {PackType(ActPack(a))} • PTATNo((PackType(pk),t_{nk})) = #((im $ActPack^{-1}$) {pk}) ∩ (im $ActType^{-1}$) {t} ∨ t_{pk} = t

PackageClass1.PackEnd(pk)

Types

- pk is a package, t is an activity type

Preconditions

- The activity to be started is associated with a package
- pk is the package the activity is associated with
- t is the activity type of the activity
- pk has not been secured
- If the package has not yet started then invoke **PackageClass1.PackStart**
- Invoke **PackageClass1.Start**

Case 1

- The package type of pk is in PTClass_No
- The package, pk, is one activity short of completion with respect to t
- The package is already complete with respect to all (appropriate) types other than t
- Invoke **PackageClass1.PackEnd**

PackageClass2.Tick()

PackageClass1.Tick()

∀pk: ((im $PackType^{-1}$) PTClass_Time) \ Pack_Complete •
#(im Earlier){now} ∩ (im Later){PackStart(pk)} > PT_Duration(PackType(pk)) ⇒ pk ∈ Pack_Delivered'

Preconditions

- Invoke **PackageClass1.Tick**

Postconditions

- Any package that is of a type in PTClass_Time and is not complete that started longer ago than is permitted by the function PT_Duration, becomes an element of Pack_Delivered.
- **PackageClass2.PackSecure**(pk)
- **PackageClass2.PackTerminate**(pk)
- **PackageClass2.PackRefuse**(pk)
- **PackageClass2.Book**(c,τ_b,τ_e,a_c,s)
- **PackageClass2.SuddenStart**(A_p,t_c,hcp,a_c)
- **PackageClass2.End**(a)
- **PackageClass2.Suspend**(a)
- **PackageClass2.Cancel**(a)
- **PackageClass2.Associate**(a,hcp)
- **PackageClass2.Disassociate**(hcp)
- **PackageClass2.OutCreate**(p_n,t_n,hcp,a_n)
- **PackageClass2.OutGenerate**(A_p,t_c,hcp,a_c)
- **PackageClass2.OutProceed**(a)
- **PackageClass2.OutComplete**(a)
- **PackageClass2.PatReg**(p)
- **PackageClass2.PatDereg**(p)
- **PackageClass2.PatArrive**(p)
- **PackageClass2.PatDepart**(p)
- **PackageClass2.PatJoin**(a,p)
- **PackageClass2.PatLeave**(p)